

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-284995

(43)Date of publication of application : 13.10.2000

(51)Int.Cl.

G06F 12/00

(21)Application number : 11-087457

(71)Applicant : FUJITSU LTD

(22)Date of filing : 30.03.1999

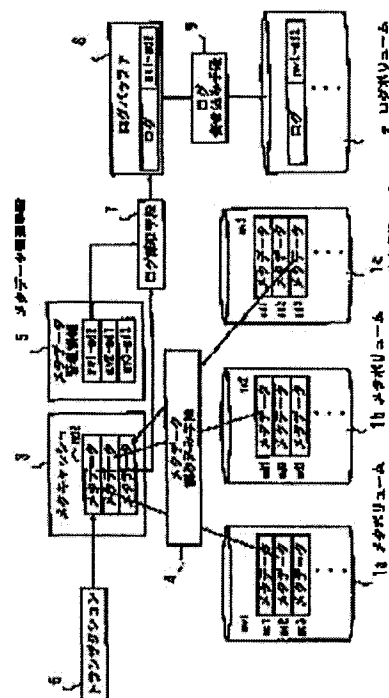
(72)Inventor : KOSEKI MICHIIHIKO
YOKOYAMA MAMORU
WASHIMI MASASHI
YAMAGUCHI SATORU
TANIWAKI SADAYOSHI
HAMANAKA SEISHIRO

(54) DATA PROCESSOR AND RECORDING MEDIUM

(57)Abstract:

PROBLEM TO BE SOLVED: To attain efficient file system restoration processing.

SOLUTION: When meta data are updated by a transaction 6, necessary meta data in meta volumes 1a-1c are read in a cache 3 by a meta data reading means 4. Which volume the read meta data are read from is managed by a meta data managing means 5. In this case, when the contents of the meta data are updated by the transaction 6, the updated meta data are adopted as a log by a log collecting means 7. In this case, the identification information of the meta volumes 1a-1c in which the meta data are stored is collected. The collected information is held in a log buffer 8. Then, the information in the log buffer 8 is written in a log volume 2 by a log writing means 9.



(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2000-284995

(P2000-284995A)

(43) 公開日 平成12年10月13日 (2000. 10. 13)

(51) Int.Cl.⁷

G 0 6 F 12/00

識別記号

5 1 4

5 3 1

F I

G 0 6 F 12/00

テ-マ-ト (参考)

5 1 4 A

5 B 0 8 2

5 3 1 J

審査請求 未請求 請求項の数21 O L (全 81 頁)

(21) 出願番号

特願平11-87457

(22) 出願日

平成11年3月30日 (1999. 3. 30)

(71) 出願人 000005223

富士通株式会社

神奈川県川崎市中原区上小田中4丁目1番
1号

(72) 発明者 小関 道彦

神奈川県川崎市中原区上小田中4丁目1番
1号 富士通株式会社内

(72) 発明者 横山 衛

神奈川県川崎市中原区上小田中4丁目1番
1号 富士通株式会社内

(74) 代理人 100092152

弁理士 服部 毅巖

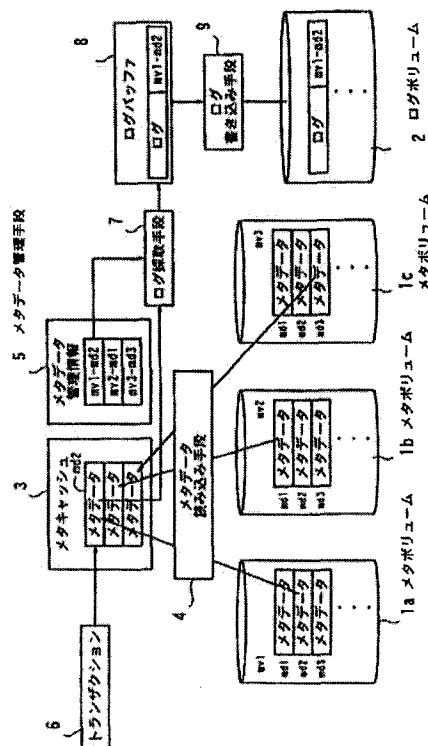
最終頁に続く

(54) 【発明の名称】 データ処理装置及び記録媒体

(57) 【要約】

【課題】 ファイルシステム修復処理の効率化を図る。

【解決手段】 トランザクション6がメタデータの更新をする際には、まず、メタデータ読み込み手段4によりメタボリューム1a~1c内の必要なメタデータがメタキャッシュ3に読み込まれる。その際、読み込んだメタデータがどのメタボリュームから読み込まれたものであるのかが、メタデータ管理手段5によって管理される。ここで、トランザクション6がメタデータの内容を更新すると、ログ採取手段7によって更新後のメタデータがログとして採取される。この際、メタデータが格納されていたメタボリューム1a~1cの識別情報をも採取する。採取した情報は、ログバッファ8に保持される。そして、ログ書き込み手段9により、ログバッファ8内の情報がログボリューム2に書き込まれる。



【特許請求の範囲】

【請求項1】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
 ファイルを管理するメタデータを記憶するための二次記憶装置である複数のメタボリュームと、
 メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、
 メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
 メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
 前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、
 前記メタキャッシュに読み込まれたメタデータが格納されていたメタボリュームの識別情報を管理するメタデータ管理手段と、
 前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取するログ採取手段と、
 前記ログ採取手段が採取した情報を保持するログバッファと、
 前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、
 を有することを特徴とするデータ処理装置。

【請求項2】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
 ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
 メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、
 メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
 メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
 前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、
 前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、
 前記ログ採取手段が採取した情報を保持するログバッファと、
 前記ログボリューム内の領域を定期的に循環するようにして、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、
 前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段と、
 前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されて

いないメタデータに対応する前記ログボリューム内のログを、有効なログとして指定する有効範囲監視手段と、
 ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログの中で、前記有効範囲監視手段により有効なログとして指定されているログのみを用いて、前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、
 を有することを特徴とするデータ処理装置。

【請求項3】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
 ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
 メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、
 メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
 メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
 前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、
 前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、
 前記ログ採取手段が採取した情報を保持するログバッファと、
 前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、
 前記ログボリュームに格納されたログを用いて前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、
 前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する初期シーケンス番号記憶手段と、
 前記ログ書き込み手段がログの書き込みを行う際に、シーケンス番号を昇順で採番し、採番したシーケンス番号を書き込むべきログに付与しており、前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した直後には、前記初期シーケンス番号記憶手段に格納されたシーケンス番号を基準として採番するシーケンス番号採番手段と、
 を有することを特徴とするデータ処理装置。

【請求項4】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
 ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
 メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
 メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み

込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、
前記トランザクションの種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、前記メタキャッシュ内で変更されたメタデータの最終形態のみをログとして採取するログ採取手段と、
を有することを特徴とするデータ処理装置。

【請求項5】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
メタデータに対する割り当てを管理するための割り当て管理情報を複数の領域に分割して保持する割り当て管理情報保持手段と、
前記割り当て管理情報保持手段内の割り当て管理情報の一部領域の複製を生成し、獲得操作管理情報とするとともに、前記獲得操作管理情報内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を前記獲得操作管理情報とする獲得操作管理情報生成手段と、
メタデータの獲得及び解放要求を出力するトランザクションと、
前記トランザクションによりメタデータの獲得要求が出された場合には、前記獲得操作管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記獲得操作管理情報と前記割り当て管理情報との内容を変更するメタデータ獲得手段と、
前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、
を有することを特徴とするデータ処理装置。

【請求項6】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
メタデータに対する割り当てを管理するための割り当て管理情報を保持する割り当て管理情報保持手段と、
メタデータの獲得及び解放要求を出力するトランザクションと、
前記トランザクションによりメタデータの獲得要求が出された場合には、前記割り当て管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記割り当て管理情報の内容を変更するメタデータ獲得手段と、
前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、
前記割り当て管理情報内の前記メタデータ獲得手段及び前記メタデータ解放手段によって変更された部分の情報をログとして採取するログ採取手段と、
を有することを特徴とするデータ処理装置。

【請求項7】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、

10 前記メタキャッシュ内に読み込まれたメタデータの内容を更新する複数のトランザクションと、
ログをトランザクション毎に保持する、サイズの異なる複数のログバッファと、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて前記ログバッファに格納するログ採取手段と、
を有することを特徴とするデータ処理装置。

【請求項8】 前記ログ採取手段は、最初にログを格納する場合には、トランザクションの内容によって予想される処理に適した大きさの前記ログバッファに格納し、
20 格納対象となる前記ログバッファの記憶容量が不足してきたら、より大きな記憶容量のログバッファへログを移し替え、以後、より大きな記憶容量のログバッファをログの格納対象とすることを特徴とする請求項7記載のデータ処理装置。

【請求項9】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
30 メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、
ログをトランザクション毎に保持するログバッファと、
40 前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記ログバッファに格納するログ採取手段と、
前記トランザクションが終了した場合に前記ログバッファの内容を前記ログボリュームに書き込むとともに、前記トランザクションによるログが前記ログバッファ内に格納しきれない場合には、前記ログバッファ内のデータを完結したログに加工し、中間ログとして前記ログボリューム内に格納するログ書き込み手段と、
を有することを特徴とするデータ処理装置。

50 【請求項10】 前記ログ書き込み手段は、前記中間ロ

グに対して、トランザクションを実行するのに必要とされたパラメタに関する情報を付加し、ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログを用いて、前記メタボリューム内のメタデータの不整合を修正するとともに、前記中間ログを発見すると、前記中間ログに含まれたパラメタを用いて、前記トランザクションを再実行させるファイルシステム復元手段をさらに有することを特徴とする請求項9記載のデータ処理装置。

【請求項11】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新する、複数同時実行可能なトランザクションと、前記トランザクションからの開始要求を受け付けると、ログ採取に関するシステムの動作状況を判断し、前記トランザクションの受け入れ可否を判断するトランザクション受け入れ判断手段と、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、を有することを特徴とするデータ処理装置。

【請求項12】 前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段と、前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応する前記ログボリューム内のログを、有効なログとして指定する有効範囲監視手段とをさらに有し、前記トランザクション受け入れ判断手段は、前記有効範囲監視手段によって有効なログとされたログがログボリューム中に占める割合が一定値以上である間は、前記トランザクションの受け入れを拒絶することを特徴とする請求項11記載のデータ処理装置。

【請求項13】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、ファイルを管理するメタデータを記憶するための二次記憶装置であ

る複数のメタボリューム、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、前記メタキャッシュに読み込まれたメタデータが格納されていたメタボリュームの識別情報を管理するメタデータ管理手段、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取するログ採取手段、前記ログ採取手段が採取した情報を保持するログバッファ、前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段、としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項14】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段、前記ログ採取手段が採取した情報を保持するログバッファ、前記ログボリューム内の領域を定期的に循環するようにして、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段、前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段、前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応する前記ログボリューム内のロ

グを、有効なログとして指定する有効範囲監視手段、
 ファイルシステム復元要求を受け取ると、前記ログボリ
 ユームに格納されたログの中で、前記有効範囲監視手段
 により有効なログとして指定されているログのみを用い
 て、前記メタボリウム内のメタデータの不整合を修正
 するファイルシステム復元手段、
 としてコンピュータを機能させることを特徴とするファ
 イル管理プログラムを記録したコンピュータ読み取り可
 能な記録媒体。

【請求項15】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
 ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリウム、
 メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリウム、
 メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
 メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
 前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
 前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段、
 前記ログ採取手段が採取した情報を保持するログバッファ、
 前記ログバッファが保持する情報を前記ログボリウムに格納するログ書き込み手段、
 前記ログボリウムに格納されたログを用いて前記メタボリウム内のメタデータの不整合を修正するファイルシステム復元手段、
 前記ファイルシステム復元手段が前記メタボリウム内のメタデータの不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する初期シーケンス番号記憶手段、
 前記ログ書き込み手段がログの書き込みを行う際に、システムの使用可能年数以上使い続けることができる値を最大値としたシーケンス番号を昇順で採番し、書き込むべきログに付与しており、前記ファイルシステム復元手段が前記メタボリウム内のメタデータの不整合を修正した直後には、前記初期シーケンス番号記憶手段に格納されたシーケンス番号から昇順で採番するシーケンス番号採番手段、
 としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項16】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、

ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリウム、
 メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
 メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
 前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
 前記トランザクションの種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、前記メタキャッシュ内で変更されたメタデータの最終形態のみをログとして採取するログ採取手段、
 としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

10

20

30

40

50

【請求項17】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
 メタデータに対する割り当てを管理するための割り当て管理情報を複数の領域に分割して保持する割り当て管理情報保持手段、
 前記割り当て管理情報保持手段内の割り当て管理情報の一部領域の複製を生成し、獲得操作管理情報とするとともに、前記獲得操作管理情報内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を前記獲得操作管理情報とする獲得操作管理情報生成手段、
 メタデータの獲得及び解放要求を出力するトランザクション、
 前記トランザクションによりメタデータの獲得要求が出された場合には、前記獲得操作管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記獲得操作管理情報と前記割り当て管理情報との内容を変更するメタデータ獲得手段、
 前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段、
 としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項18】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
 メタデータに対する割り当てを管理するための割り当て管理情報を保持する割り当て管理情報保持手段、
 メタデータの獲得及び解放要求を出力するトランザクション、
 前記トランザクションによりメタデータの獲得要求が出

された場合には、前記割り当て管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記割り当て管理情報の内容を変更するメタデータ獲得手段、

前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段、

前記割り当て管理情報内の前記メタデータ獲得手段及び前記メタデータ解放手段によって変更された部分の情報をログとして採取するログ採取手段、

としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項19】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、

ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、

メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、

メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、

前記メタキャッシュ内に読み込まれたメタデータの内容を更新する複数のトランザクション、

ログをトランザクション毎に保持する、サイズの異なる複数のログバッファ、

前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて前記ログバッファに格納するログ採取手段、

としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項20】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、

ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、

メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、

メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、

メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、

前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、

ログをトランザクション毎に保持するログバッファ、

前記メタキャッシュ内で変更されたメタデータの内容を

ログとして採取し、前記ログバッファに格納するログ採取手段、

前記トランザクションが終了した場合に前記ログバッファの内容を前記ログボリュームに書き込むとともに、前記トランザクションによるログが前記ログバッファ内に格納しきれない場合には、前記ログバッファ内のデータを完結したログに加工し、中間ログとして前記ログボリューム内に格納するログ書き込み手段、

としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項21】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、

ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、

メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、

メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、

メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、

前記メタキャッシュ内に読み込まれたメタデータの内容を更新する、複数同時実行可能なトランザクション、

前記トランザクションからの開始要求を受け付けると、ログ採取に関するシステムの動作状況を判断し、前記トランザクションの受け入れ可否を判断するトランザクション受け入れ判断手段、

前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段、

前記ログ採取手段が採取した情報を保持するログバッファ、

前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段、

としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】 本発明はファイルシステムの修正機能を有するデータ処理装置及び記録媒体に関し、特にログを用いてファイルシステムの不整合の修正を行うデータ処理装置及び記録媒体に関する。

【0002】

【従来の技術】 コンピュータシステムを運用していると、何らかの理由によりシステムがダウンする場合がある。突然のシステムダウンが発生すると、ファイルシステムの不整合が生じる。そこで、システムダウン後のブート時には、従来であればファイルシステム全体を走査

10

20

30

40

50

し、その矛盾点の検出を行う。矛盾点が発見されたら、場合に応じた変更をファイルシステムに加えることによって、ファイルシステムの整合性を回復していた。ところが、ファイルシステム全体を走査するには多くの時間が必要であり、結果としてシステムダウン後の復旧の遅れを招いていた。

【0003】そこで近代のUNIXオペレーティングシステム(OS)のような多くのコンピュータOSのファイルシステムでは、ファイルシステムオペレーションにおいてファイルシステムに保存されているデータが更新される度に、その更新情報をログ(ジャーナル)として採取している。ファイルシステムオペレーション時に更新情報のログ採取を行っておくことにより、システムダウン後のブート時のファイルシステム整合性チェックのフェーズでは、残されたログを順次走査し、対応する領域へアップデートすることによって、ファイルシステムの整合性が保証される。その結果、システムのダウン時間が短縮される。

【0004】

【発明が解決しようとする課題】しかし、ログ機構を導入することにより、次に挙げる問題が生じていた。

(1) 第1の問題点

ファイルを管理するメタデータ(二次記憶装置に格納されたファイルを管理するためのデータ)は、ファイルシステムオペレーション時に二次記憶装置からメモリへ読み込まれ、メモリ内において操作される。その後、所定のタイミングで二次記憶装置へとその更新内容が反映される。ログ機構の導入時には、その二次記憶装置への反映前に更新内容をログ専用の二次記憶装置へと記録する必要がある。

【0005】しかし、大規模ファイルシステムに対応するために、複数の二次記憶装置がメタデータに割り当てられ、異なる二次記憶装置に割り当てられたメタデータを1つのファイルシステムオペレーションが操作する場合がある。このファイルシステムオペレーションのログを採取する際に、メモリ内のメタデータだけをログとして記録していたのでは、残されたログからメタデータ毎に異なる二次記憶装置を検索するのに時間を消費し、ログ機構導入による最大のメリットである、ファイルシステム復元の時間短縮に悪い影響を与える。

【0006】(2) 第2の問題点

同様にファイルシステム復元の時間短縮に悪い影響を与える要因として、有限なサイズしかないログ専用の二次記憶装置全体をファイルシステム復元時に探索することが考えられる。

【0007】ログ機構の導入はファイルシステムオペレーションを細分化したトランザクションの完了を常に保証しつつ動作するため、ログ機構を導入した多くのファイルシステムはトランザクションにシーケンス番号を与え、ファイルシステム復元時にはシーケンス番号をもと

に最古のトランザクションを同定する。そして、最古のトランザクションからログプレイと呼ばれるログを用いたファイルシステム復元操作を開始する。

【0008】ここで、有限サイズのログ専用の二次記憶装置内には、ファイルシステムの整合性を復元するために欠かせないログが確かに存在する可能性があるが、有限なサイズを有効利用するためにログ専用の二次記憶装置は過去のログを上書きしてサイクリックに利用しなければならない。このような処理を行うには、ある程度以上古いログが必ず不必要となっていることが前提となる。従って、多くのログの内容は既に不必要となっている。すなわち、ログ専用の二次記憶装置全体を探索あるいは反映するのは非効率的である。

【0009】(3) 第3の問題点

最古のトランザクションを特定するためのシーケンス番号は単調増加であることが要求され、運用途中にオーバーフローによってゼロに戻されてしまうことは許されない。これを回避するために、ファイルシステム復元作業終了時、またはオーバーフロー直前にログ専用の二次記憶装置全体をゼロクリアし、再度シーケンス番号ゼロから順にトランザクションを処理するのが一般的である。しかし、ログ専用の二次記憶装置をゼロクリアするためには多くの時間が必要となる。少なくともシステムの運用を一時停止する必要があり、サーバ装置などによるサービスの提供を停止せざるを得なくなってしまう。

【0010】以上の(1)～(3)の課題はファイルシステム復元時の問題であるが、ログ機構を導入することは通常の運用時にも問題を引き起こす可能性を持っている。ログ機構は、メモリ上にログをスプールする手法や、ログ専用の二次記憶装置として用いられるディスクの特性を考慮したシーケンシャルアクセスなど、高速化のための条件は整えられているが、ログの採取の仕方を熟考しなければ、ログ採取に伴う性能劣化は非常に大きいものとなりうる。

【0011】(4) 第4の問題点

単一のトランザクション内で、同一データを複数回更新することは度々あるが、その都度、その同一データに対するログを採取したのではメモリが不足し、二次記憶装置へのI/O量が増加する。

40 【0012】(5) 第5の問題点

ログ機構の導入はトランザクションの順序性を保証し、終了したトランザクションのログを順に採取することを要求するために、あるトランザクションが操作したログ対象データを他のトランザクションが操作することが一般的に不可能な状況となりうる。ここで、個々のファイルの内容を対象とするトランザクションについては、ファイル単位に排他制御を行うことによって、複数のトランザクションが並列実行することは比較的容易である。しかしながら、個々のファイルによらないもの、特に領域の割り当て情報を操作する場合には、並列実行がきわ

めて難しくなる。

【0013】領域の獲得・解放処理が並列に動作する場合を考えると、それぞれが獲得、解放のログを採取する。同一の管理情報（ここでは、ビットマップを考える）によって管理される領域の獲得・解放処理であった場合には、同一の管理情報が、解放された状態、獲得された状態でログに記録されることになる。

【0014】ファイルシステムの復元処理を行わなければならないようなシステムダウンが発生するタイミングによっては、このように別トランザクションの更新情報を含むログの採取方式では様々な問題が生じる。

【0015】Aというトランザクションが解放処理、Bというトランザクションが獲得処理を行う場合を考える。ここで、トランザクションAの解放処理はトランザクションBの獲得処理よりも先に行われ、かつ、トランザクションAはトランザクションBよりも後に終わる場合を例に挙げる。

【0016】このとき、トランザクションAが開放した領域をトランザクションBが獲得してしまう場合が考えられる。トランザクションBが先に終了することから、残されたログには、まず獲得処理が記録され、次に解放処理が記録される。

【0017】上記の場合のトランザクションBのログだけが記録されており、それを復元に用いた場合には、本来行われたはずである解放処理の記録が残されていないことから、該当領域が二重に獲得された状態となってしまう。トランザクションAのログまで記録されており、復元に用いられると、トランザクションBが利用している領域がトランザクションAの解放処理のログによって解放されている状態となってしまう。いずれも本来の状態とは異なっており、避けなければならない。しかしながら、これを回避するために並列実行を制限することは、マルチタスクを実現したOS上のファイルシステムの速度性能の低下に与える影響が非常に大きい。

【0018】（6）第6の問題点

既に述べたようにログ機構の第一の目的としてファイルシステムを復元するために費やす時間の短縮が挙げられるが、そのためにトランザクションの独立性を疎かにしたり、中途半端な状態での整合性回復によりあたかも正常に動作しているかのように振る舞うファイルシステムが多く見受けられる。

【0019】従来のメタデータ管理方式のように、ログを記録するためのメモリ空間をファイルシステム全体で1つのキャッシュメモリを共有していたのではトランザクション毎の独立性を保つのが難しく、他のトランザクションによる更新情報が1つのトランザクションのログとして記録されてしまう可能性が高い。特にファイルに対する排他で制御しきれない割り当て管理情報については前述の通りである。

【0020】（7）第7の問題点

トランザクション毎に必要なとするログのサイズが異なることから、複数のログバッファとしてログを記録するためのメモリをトランザクション毎に割り当て、管理する場合に、全てのメモリサイズが同一である必要はない。例えば、ファイルの参照時刻を更新するだけのトランザクションが残すログのサイズは大変小さく、巨大なデータ書き込み要求のトランザクションは必然的にそれだけ大きいログサイズとなる。

【0021】（8）第8の問題点

トランザクション毎に残すログサイズの違いを考慮して、限られたメモリ空間の有効利用を試みても、キャッシュメモリサイズは残されるログサイズに比較して、やはり小さい。

【0022】（9）第9の問題点

「第8の問題点」の解決策として、1つのトランザクションを分割し、中途の状態のログを出力することが考えられる。しかし、一般にファイルを管理するメタデータは、トランザクションが中途の状態ではやはり中途の状態であり、そのままログに記録したところで、そのログを用いて復元されるファイルシステムは中途の状態にかなり得ない。これではオペレーションのセマンティクスを保証した復元とはなり得ない。

【0023】（10）第10の問題点

「第9の問題点」で示した中途の状態でのトランザクションの中断はファイルシステムにとって危険な状態といえる。ログ機構の導入は、採取したログをログボリュームに反映するまでは該当メタデータもキャッシュメモリから追い出せないことを意味する。ログがキャッシュメモリに入りきらないほど巨大となっているときには、メタデータを管理するキャッシュメモリの利用率も高くなっていることが考えられる。ログを出力するまでメタデータをキャッシュメモリから追い出せないため、このような状態で多くのトランザクションが並列実行されると、メモリ枯渇によるハングアップ状態に陥ることも考えられる。

【0024】（11）第11の問題点

第10の問題点と同様の資源枯渇はログ記録用の二次記憶装置についても言える。キャッシュメモリに比較すれば巨大な二次記憶装置についても、メタデータ記録用の二次記憶装置へのI/Oを削減するために、多くのログを有効なログとして残しておけば、それは利用可能な領域の減少を引き起こす。その際に多数のトランザクションの並列実行を許すことはメタデータキャッシュ枯渇、ログキャッシュ枯渇、ログ記録用二次記憶装置枯渇、などを誘発する可能性がある。

【0025】本発明はこのような点に鑑みてなされたものであり、ファイルシステム修復処理の効率化を図ったデータ処理装置を提供することを目的とする。

【0026】

50 【課題を解決するための手段】本発明では上記課題を解

決するための第1の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置である複数のメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュに読み込まれたメタデータが格納されていたメタボリュームの識別情報を管理するメタデータ管理手段と、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、を有することを特徴とするデータ処理装置が提供される。

【0027】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。その際、読み込んだメタデータがどのメタボリュームから読み込まれたものであるのかが、メタデータ管理手段によって管理される。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。この際、メタデータが格納されていたメタボリュームの識別情報をも採取する。採取した情報は、ログバッファに保持される。そして、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。

【0028】また、上記課題を解決する第2の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログボリューム内の領域を定期的に循環するようにして、前記ログバッファが保持する情報を前記ログボリュームに格納

するログ書き込み手段と、前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段と、前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応する前記ログボリューム内のログを、有効なログとして指定する有効範囲監視手段と、ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログの中で、前記有効範囲監視手段により有効なログとして指定されているログのみを用いて、前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、を有することを特徴とするデータ処理装置が提供される。

【0029】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファに保持される。そして、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。その後、メタデータ書き込み手段により、メタキャッシュ内のメタデータがメタボリューム内に格納される。その書き込み動作は、有効範囲監視手段で監視されており、変更内容がメタボリュームに反映されていないメタデータに対応するログボリューム内のログが、有効なログとして指定される。そして、ファイルシステム復元要求が出されると、ファイルシステム復元手段により、ログボリュームに格納されたログの中で、有効範囲監視手段により有効なログとして指定されているログのみを用いて、メタボリューム内のメタデータの不整合が修正される。

【0030】また、上記課題を解決する第3の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、前記ログボリュームに格納されたログを用いて前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、前記ファイルシステム復元手段が前記メタボリューム内のメタデータ

の不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する初期シーケンス番号記憶手段と、前記ログ書き込み手段がログの書き込みを行う際に、シーケンス番号を昇順で採番し、採番したシーケンス番号を書き込むべきログに付与しており、前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した直後には、前記初期シーケンス番号記憶手段に格納されたシーケンス番号を基準として採番するシーケンス番号採番手段と、を有することを特徴とするデータ処理装置が提供される。

【0031】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファに保持される。そして、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。その際、シーケンス番号採番手段により、シーケンス番号が昇順で採番され、書き込むべきログに付与される。ファイルシステムに不整合が発生すると、ファイルシステム復元手段により、ログボリュームに残されたログを用いてメタデータの不整合が修正される。このとき、修正に用いられた最後のログのシーケンス番号が初期シーケンス番号記憶手段に記憶される。その後、ログ書き込み手段によりログバッファ内の情報がログボリュームに書き込まれると、シーケンス番号採番手段により、初期シーケンス番号記憶手段に格納されたシーケンス番号を基準としてシーケンス番号が採番され、書き込むべきログに付与される。

【0032】また、上記課題を解決する第4の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイル进行管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記トランザクションの種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、前記メタキャッシュ内で変更されたメタデータの最終形態のみをログとして採取するログ採取手段と、を有することを特徴とするデータ処理装置が提供される。

【0033】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。こ

で、トランザクションがメタデータの内容を更新する。すると、ログ採取手段により、トランザクションの種別が判断され、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、メタキャッシュ内で変更されたメタデータの最終形態のみがログとして採取される。

【0034】また、上記課題を解決する第5の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、メタデータに対する割り当てを管理するための割り当て管理情報を複数の領域に分割して保持する割り当て管理情報保持手段と、前記割り当て管理情報保持手段内の割り当て管理情報の一部領域の複製を生成し、獲得操作管理情報とするとともに、前記獲得操作管理情報内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を前記獲得操作管理情報とする獲得操作管理情報生成手段と、メタデータの獲得及び解放要求を出力するトランザクションと、前記トランザクションによりメタデータの獲得要求が出された場合には、前記獲得操作管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記獲得操作管理情報と前記割り当て管理情報との内容を変更するメタデータ獲得手段と、前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、を有することを特徴とするデータ処理装置が提供される。

【0035】このようなデータ処理装置によれば、獲得操作管理情報生成手段により、割り当て管理情報の一部領域の複製が生成され、獲得操作管理情報とされる。トランザクションにより獲得要求があると、メタデータ獲得手段により、獲得操作管理情報内の未獲得のメタデータが獲得される。すると、獲得操作管理情報と割り当て管理情報との内容が更新される。また、トランザクションよりメタデータの解放要求があると、メタデータ解放手段により該当するメタデータの解放処理が行われる。この際、割り当て管理情報の内容のみが更新される。ここで、獲得操作管理情報内に未獲得のメタデータがなくなると、割り当て管理情報内の別の領域の複製が獲得操作管理情報とされる。

【0036】また、上記課題を解決する第6の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、メタデータに対する割り当てを管理するための割り当て管理情報を保持する割り当て管理情報保持手段と、メタデータの獲得及び解放要求を出力するトランザクションと、前記トランザクションによりメタデータの獲得要求が出された場合には、前記割り当て管理情報保の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記割り当て管理情報の内容を変更するメタデータ獲得手段

と、前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、前記割り当て管理情報内の前記メタデータ獲得手段及び前記メタデータ解放手段によって変更された部分の情報をログとして採取するログ採取手段と、を有することを特徴とするデータ処理装置が提供される。

【0037】このようなデータ処理装置によれば、トランザクションからメタデータの獲得要求が出されると、メタデータ獲得手段によって未獲得のメタデータの1つが割り当て管理情報内から獲得され、そのメタデータが獲得済みの状態とされる。また、トランザクションからメタデータの解放要求が出されると、メタデータ解放手段によって該当するメタデータが未獲得の状態に変更される。そして、ログ採取手段により、割り当て管理情報内のメタデータ獲得手段及びメタデータ解放手段によって変更された部分の情報がログとして採取される。

【0038】また、上記課題を解決する第7の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新する複数のトランザクションと、ログをトランザクション毎に保持する、サイズの異なる複数のログバッファと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて前記ログバッファに格納するログ採取手段と、を有することを特徴とするデータ処理装置が提供される。

【0039】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新する。すると、ログ採取手段により、メタキャッシュ内で変更されたメタデータがログとして採取され、トランザクション毎のログバッファに保持される。

【0040】また、上記課題を解決する第8の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ

読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、ログをトランザクション毎に保持するログバッファと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記ログバッファに格納するログ採取手段と、前記トランザクションが終了した場合に前記ログバッファの内容を前記ログボリュームに書き込むとともに、前記トランザクションによるログが前記ログバッファ内に格納しきれない場合には、前記ログバッファ内のデータを完結したログに加工し、中間ログとして前記ログボリューム内に格納するログ書き込み手段と、を有することを特徴とするデータ処理装置が提供される。

【0041】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファに保持される。そして、ログバッファに格納しきれなくなるか、トランザクションが終了すると、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。ログバッファに格納しきれなくなった場合には、ログバッファの内容を完結したログに加工され、中間ログとしてログボリュームに格納される。

【0042】また、上記課題を解決する第9の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新する、複数同時実行可能なトランザクションと、前記トランザクションからの開始要求を受け付けると、ログ採取に関するシステムの動作状況を判断し、前記トランザクションの受け入れ可否を判断するトランザクション受け入れ判断手段と、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、を有することを特徴とするデータ処理装置が提供される。

【0043】このようなデータ処理装置によれば、トランザクションから開始要求が出されると、トランザクション受け入れ制限手段が受け入れの可否を判断する。その後、メタデータ書き込み手段によるメタデータの書き

10

20

30

40

50

込みが進み、有効なログの割合が減少したら、その時点でトランザクションの開始要求を許可する。

【0044】

【発明の実施の形態】以下、本発明の実施の形態を図面を参照して説明する。図1は、第1の発明の原理構成図である。この第1の発明は、複数の二次記憶装置がメタデータに割り当てられている場合におけるファイルシステム不整合修復時間の短縮を図るものである。

【0045】このデータ処理装置には、二次記憶装置としてメタボリューム1a、1b、1cとログボリューム2とが設けられている。各メタボリューム1a、1b、1cには、ファイルを管理するためのメタデータが記憶されている。また、ログボリューム2には、メタデータの更新結果であるログが記憶されている。また、主記憶装置内にはメタキャッシュ3が設けられている。メタキャッシュ3は、メタデータを記憶するための主記憶装置内の記憶領域である。メタデータ読み込み手段4は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ3へと読み込む。メタデータ管理手段5は、メタキャッシュ3に読み込まれたメタデータが格納されていたメタボリューム1a、1b、1cの識別情報を管理する。トランザクション6は、メタキャッシュ3内に読み込まれたメタデータの内容を更新する。ログ採取手段7は、メタキャッシュ3内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリューム1a、1b、1cの識別情報を採取する。ログバッファ8は、ログ採取手段7が採取した情報を保持する。ログ書き込み手段9は、ログバッファ8が保持する情報を、適宜ログボリューム2に格納する。

【0046】このようなデータ処理装置よれば、トランザクション6がメタデータの更新をする際には、まず、メタデータ読み込み手段4によりメタボリューム1a～1c内の必要なメタデータがメタキャッシュ3に読み込まれる。その際、読み込んだメタデータがどのメタボリュームから読み込まれたものであるのかが、メタデータ管理手段5によって管理される。ここで、トランザクション6がメタデータの内容を更新すると、ログ採取手段7によって更新後のメタデータがログとして採取される。この際、メタデータが格納されていたメタボリューム1a～1cの識別情報をも採取する。採取した情報は、ログバッファ8に保持される。そして、ログ書き込み手段9により、ログバッファ8内の情報がログボリューム2に書き込まれる。

【0047】これにより、ログボリューム2に保持されたログがどのメタボリューム1a～1cのメタデータに関するログであるのかを管理することができる。その結果、複数のメタボリューム1a～1cにメタデータが格納されていても、ファイルシステムの不整合を修正することが可能となる。

【0048】図2は、第2の発明の原理構成図である。第2の発明は、ログの有効範囲を監視することで、ファイルシステム復元時の効率化を図ったものである。第2の発明は、以下のような要素で構成される。

【0049】メタボリューム11は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム12は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ13は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段14は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ13へと読み込む。トランザクション15は、メタキャッシュ13内に読み込まれたメタデータの内容を更新する。ログ採取手段16は、メタキャッシュ13内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取する。ログバッファ17は、ログ採取手段16が採取した情報を保持する。ログ書き込み手段18は、ログボリューム12内の領域を定期的に循環するようにして、ログバッファ17が保持する情報をログボリューム12に格納する。メタデータ書き込み手段19は、メタキャッシュ13内のメタデータをメタボリューム11内に格納する。有効範囲監視手段20は、メタデータ書き込み手段19による書き込み動作を監視しており、変更内容がメタボリューム11に反映されていないメタデータに対応するログボリューム12内のログを、有効なログとして指定する。ファイルシステム復元手段21は、ファイルシステム復元要求を受け取ると、ログボリューム12に格納されたログの中で、有効範囲監視手段20により有効なログとして指定されているログのみを用いて、メタボリューム11内のメタデータの不整合を修正する。なお、有効範囲監視手段20による有効なログの指定方法としては、例えば、有効範囲を示す情報を不揮発性の記録媒体（ログボリューム12等）に記録することができる。この場合、ファイルシステム復元手段21は、有効範囲が記録された不揮発性の記録媒体内の情報を読みとることで、有効なログと指定されているログを認識できる。

【0050】このようなデータ処理装置によれば、トランザクション15がメタデータの更新をする際には、まず、メタデータ読み込み手段14によりメタボリューム11内の必要なメタデータがメタキャッシュ13に読み込まれる。ここで、トランザクション15がメタデータの内容を更新すると、ログ採取手段16によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファ17に保持される。そして、ログ書き込み手段18により、ログバッファ17内の情報がログボリューム12に書き込まれる。その後、メタデータ書き込み手段19により、メタキャッシュ13内のメタデータがメタボリューム11内に格納される。その書き込

み動作は、有効範囲監視手段 20 で監視されており、変更内容がメタボリューム 11 に反映されていないメタデータに対応するログボリューム 12 内のログが、有効なログとして指定される。そして、ファイルシステム復元要求が出されると、ファイルシステム復元手段 21 により、ログボリューム 12 に格納されたログの中で、有効範囲監視手段 20 により有効なログとして指定されているログのみを用いて、メタボリューム 11 内のメタデータの不整合が修正される。

【0051】これにより、ファイルシステムを復元する際には、ログボリューム 12 内の有効なログのみを用いて効率よく復元処理を行うことが可能となる。図 3 は、第 3 の発明の原理構成図である。第 3 の発明は、ログに付加するシーケンス番号のデータサイズを十分大きなものとし、シーケンス番号を常に昇順で使い続けることができる（ゼロクリアが不要となる）ようにしたものである。第 3 の実施の形態は、以下のような要素で構成される。

【0052】メタボリューム 31 は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム 32 は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ 33 は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段 34 は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ 33 へと読み込む。トランザクション 35 は、メタキャッシュ 33 内に読み込まれたメタデータの内容を更新する。ログ採取手段 36 は、メタキャッシュ 33 内で変更されたメタデータの内容をログとして採取する。ログバッファ 37 は、ログ採取手段 36 が採取した情報を保持する。ログ書き込み手段 38 は、ログバッファが保持する情報を前記ログボリュームに格納する。ファイルシステム復元手段 39 は、ログボリューム 32 に格納されたログを用いてメタボリューム 31 内のメタデータの不整合を修正する。初期シーケンス番号記憶手段 30a は、ファイルシステム復元手段 39 がメタボリューム 31 内のメタデータの不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する。シーケンス番号採番手段 30b は、ログ書き込み手段 38 がログの書き込みを行う際に、シーケンス番号を昇順で採番し、採番したシーケンス番号を書き込むべきログに付与しており、ファイルシステム復元手段 39 がメタボリューム 31 内のメタデータの不整合を修正した直後には、初期シーケンス番号記憶手段 30a に格納されたシーケンス番号を基準として採番する。

【0053】このようなデータ処理装置によれば、トランザクション 35 がメタデータの更新をする際には、まず、メタデータ読み込み手段 34 によりメタボリューム 31 内の必要なメタデータがメタキャッシュ 33 に読み込まれる。ここで、トランザクション 35 がメタデータ

の内容を更新すると、ログ採取手段 36 によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファ 37 に保持される。そして、ログ書き込み手段 38 により、ログバッファ 37 内の情報がログボリューム 32 に書き込まれる。その際、シーケンス番号採番手段 30b によりシーケンス番号が昇順で採番され、書き込むべきログに付与される。また、ファイルシステム復元手段 39 により、ログボリューム 32 に格納されたログを用いてメタボリューム 31 内のメタデータの不整合が修正されると、修正した時に用いられたログの最後のシーケンス番号が初期シーケンス番号記憶手段 30a に記憶される。その後、ログ書き込み手段 38 により、ログバッファ 37 内の情報がログボリューム 32 に書き込まれると、シーケンス番号採番手段 30b により、初期シーケンス番号採番手段 30a に記憶されたシーケンス番号を基準としてシーケンス番号が昇順で採番され、書き込むべきログに付与される。

【0054】これにより、既にファイルシステムの復元に用いたログと、ファイルシステム復元後に介したトランザクションのログとのシーケンス番号が重ならないことを保証することができる。その結果、システムが使用可能な期間中にログボリュームをゼロクリアする必要がなくなり、ゼロクリアに伴う処理の遅延を避けることができる。

【0055】図 4 は、第 4 の発明の原理構成図である。第 4 の発明は、同じメタデータに対して複数回更新処理が行われる場合に、最終形態のメタデータのみをログとして採取するものである。第 4 の発明は、以下のような要素で構成される。

【0056】メタボリューム 41 は、ファイルを管理するメタデータを記憶するための二次記憶装置である。メタキャッシュ 42 は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段 43 は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ 42 へと読み込む。トランザクション 44 は、メタキャッシュ 42 内に読み込まれたメタデータの内容を更新する。ログ採取手段 45 は、トランザクション 44 の種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、メタキャッシュ 42 内で変更されたメタデータの最終形態のみをログとして採取する。ログバッファ 46 は、ログ採取手段 45 が採取した情報を保持する。

【0057】このようなデータ処理装置によれば、トランザクション 44 がメタデータの更新をする際には、まず、メタデータ読み込み手段 43 によりメタボリューム 41 内の必要なメタデータがメタキャッシュ 42 に読み込まれる。ここで、トランザクション 44 がメタデータの内容を更新する。すると、ログ採取手段 45 により、トランザクション 44 の種別が判断され、メタデータの

更新を複数回行う可能性のあるメタデータ属性を予測する。予測されたメタデータ属性において、同一メタデータが何度も更新される可能性がある場合には、その属性のメタデータがメタキャッシュ42内で変更された時点ではログ採取を行わず、トランザクション44が終了した時点で、最終形態のメタデータをログとして採取する。採取した情報は、ログバッファ46に保持される。

【0058】これにより、複数回更新されたメタデータのログを更新処理の度に採取することがなくなり、メモリの効率化を図ることができるとともに、ログを書き出すときのI/Oの量も減らすことができる。

【0059】図5は、第5の発明の原理構成図である。第5の発明は、メタデータ割り当て管理情報の一部の複製を生成し、複製として生成した情報内からのみメタデータの獲得を可能とし、解放する際には、割り当て管理情報においてのみ解放された旨の情報の更新を行うことで、解放直後に別のトランザクションに獲得されるのを防止したものである。第5の発明は、以下のような要素で構成される。

【0060】割り当て管理情報保持手段51は、メタデータに対する割り当てを管理するための割り当て管理情報51aを複数の領域に分割して保持する。獲得操作管理情報生成手段52は、割り当て管理情報保持手段51内の割り当て管理情報の一部領域の複製を生成し、獲得操作管理情報51bとする。また、獲得操作管理情報51b内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を獲得操作管理情報51bとする。トランザクション53は、メタデータの獲得及び解放要求を出力する。メタデータ獲得手段54は、トランザクション53によりメタデータの獲得要求が出された場合には、獲得操作管理情報51bの中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように獲得操作管理情報51bと割り当て管理情報51aとの内容を変更する。メタデータ解放手段55は、トランザクション53によりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、割り当て管理情報の内容を変更する。

【0061】このようなデータ処理装置によれば、獲得操作管理情報生成手段52により、割り当て管理情報51aの一部領域の複製が生成され、獲得操作管理情報51bとされる。トランザクション53により獲得要求があると、メタデータ獲得手段54により、獲得操作管理情報51b内の未獲得のメタデータが獲得される。すると、獲得操作管理情報51bと割り当て管理情報51aとの内容が更新される。また、トランザクション53よりメタデータの解放要求があると、メタデータ解放手段55により該当するメタデータの解放処理が行われる。この際、割り当て管理情報51aの内容のみが更新される。ここで、獲得操作管理情報51b内に

未獲得のメタデータがなくなると、割り当て管理情報51a内の別の領域の複製が獲得操作管理情報51bとされる。

【0062】これにより、解放された旨の情報が獲得操作管理情報51bに反映されないため、解放直後のメタデータが他のトランザクションに獲得されることがなくなる。その結果、解放処理を行ったトランザクションの終了前にシステムがダウンしても、少なくとも解放前の状態のまま保全されることが保証される。

【0063】図6は、第6の発明の原理構成図である。第6の発明は、トランザクション単位に、獲得や解放に関する情報をログとして記録することで、必要なメモリ容量の削減を図るとともに、平行動作するトランザクションのログに起因して割り当て管理情報に不正な状態が発生することを防ぐものである。第6の発明は、以下のような要素で構成される。

【0064】割り当て管理情報保持手段61は、メタデータに対する割り当てを管理するための割り当て管理情報を保持する。トランザクション62は、メタデータの獲得及び解放要求を出力する。メタデータ獲得手段63は、トランザクション62によりメタデータの獲得要求が出された場合には、獲得操作管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように割り当て管理情報61aの内容を変更する。メタデータ解放手段64は、トランザクション62によりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、割り当て管理情報61aの内容を変更する。ログ採取手段65は、割り当て管理情報内のメタデータ獲得手段63及びメタデータ解放手段64によって変更された部分の情報をログとして採取する。ログバッファ66は、ログ採取手段65が採取したログを保持する。

【0065】このようなデータ処理装置によれば、トランザクション62からメタデータの獲得要求が出されると、メタデータ獲得手段63によって未獲得のメタデータの1つが割り当て管理情報61a内から獲得され、そのメタデータが獲得済みの状態とされる。また、トランザクション62からメタデータの解放要求が出されると、メタデータ解放手段64によって該当するメタデータが未獲得の状態に変更される。そして、ログ採取手段65により、割り当て管理情報61a内のメタデータ獲得手段63及びメタデータ解放手段64によって変更された部分の情報がログとして採取され、ログバッファ66に保持される。

【0066】このように、獲得、解放のログを採取する際に、トランザクションが獲得や解放を行ったという情報のみをログとして格納することで、メモリ等の領域を効率よく利用することができるとともに、他トランザクションによる獲得や解放処理の情報がログに含まれないことによって、割り当て管理情報が不正な状態となるこ

とを防ぐことができる。

【0067】図7は、第7の発明の原理構成図である。第7の発明は、ログバッファを複数設けることにより、トランザクションの独立性をいっそう高めたものである。第7の発明の構成要素は以下の通りである。

【0068】メタボリューム71は、ファイルを管理するメタデータを記憶するための二次記憶装置である。メタキャッシュ72は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段73は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ72へと読み込む。複数のトランザクション74a~74cは、メタキャッシュ72内に読み込まれたメタデータの内容を更新する。複数のログバッファ75a~75eは、ログをトランザクション毎に保持する。各ログバッファ75a~75eのサイズは一定ではなく、大きなサイズや小さなサイズが存在する。ログ採取手段76は、メタキャッシュ72内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて適したサイズのログバッファ75a~75eに格納する。例えば、最初にログを格納する場合には、トランザクションの内容によって予想される処理に適した大きさのログバッファに格納し、格納対象となるログバッファの記憶容量が不足してきたら、より大きな記憶容量のログバッファへログを移し替え、以後、より大きな記憶容量のログバッファをログの格納対象とする。

【0069】このようなデータ処理装置によれば、トランザクション74a~74cがメタデータの更新をする際には、まず、メタデータ読み込み手段73によりメタボリューム71内の必要なメタデータがメタキャッシュ72に読み込まれる。ここで、トランザクション74a~74cがメタデータの内容を更新する。すると、ログ採取手段76により、メタキャッシュ72内で変更されたメタデータがログとして採取され、適したサイズのログバッファ75a~75eに保持される。

【0070】このように、複数のログバッファに分け、トランザクション毎に1つのログバッファを使用するようにしたことで、トランザクションの独立性を保つことができる。しかも、複数のサイズのログバッファを用意し、トランザクション毎に適したサイズのログバッファを使用することにより、メモリを効率的に利用できる。

【0071】図8は、第8の発明の原理構成図である。第8の発明は、あるトランザクションのログがログバッファに入りきらない場合に、中間ログとしてログバッファの内容を書き出すものである。第8の発明の構成は以下の通りである。

【0072】メタボリューム81は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム82は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ

83は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段84は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ83へと読み込む。トランザクション85は、メタキャッシュ83内に読み込まれたメタデータの内容を更新する。ログバッファ86は、ログをトランザクション毎に保持する。ログ採取手段87は、メタキャッシュ83内で変更されたメタデータの内容をログとして採取し、ログバッファ86に格納する。ログ書き込み手段88は、トランザクション85が終了した場合にログバッファ86の内容をログボリューム82に書き込むとともに、トランザクション85によるログがログバッファ86内に格納しきれない場合には、ログバッファ86内のデータを完結したログに加工し、中間ログとしてログボリューム82内に格納する。なお、中間ログを生成する際には、中間ログに対してトランザクションを実行するのに必要とされたパラメタに関する情報を付加する。ファイルシステム復元手段89は、ファイルシステム復元要求を受け取ると、ログボリューム82に格納されたログを用いて、メタボリューム81内のメタデータの不整合を修正する。このとき、中間ログを発見すると、中間ログに含まれたパラメタを用いてトランザクションを再実行させる。

【0073】このようなデータ処理装置によれば、トランザクション85がメタデータの更新をする際には、まず、メタデータ読み込み手段84によりメタボリューム81内の必要なメタデータがメタキャッシュ83に読み込まれる。ここで、トランザクション85がメタデータの内容を更新すると、ログ採取手段87によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファ86に保持される。そして、ログバッファ86に格納しきれなくなるか、トランザクション85が終了すると、ログ書き込み手段88により、ログバッファ86内の情報がログボリューム82に書き込まれる。ログバッファ86に格納しきれなくなった場合には、ログバッファ86の内容を完結したログに加工し、中間ログとしてログボリューム82に格納する。そして、ファイルシステム復元要求が出されると、ファイルシステム復元手段89により、ログボリューム82に格納されたログを用いて、メタボリューム81内のメタデータの不整合が修正されるとともに、中間ログまで採取した段階で停止しているトランザクションが再実行される。

【0074】これにより、メタデータの更新を大量に行うトランザクションの実行中にシステムがダウンした場合には、途中の状態まで戻すことができるとともに、トランザクションが再実行されることで、トランザクションの実行後の状態へ遷移させることができる。

【0075】図9は、第9の発明の原理構成図である。トランザクションの受け入れを一定の条件によって制限

することで、メモリ枯渇等を防止するものである。第9の発明の構成は以下の通りである。

【0076】メタボリューム91は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム92は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ94は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段93は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ94へと読み込む。互いの同時実行可能な複数のトランザクション90b~90dは、メタキャッシュ94内に読み込まれたメタデータの内容を更新する。トランザクション受け入れ制限手段90aは、トランザクション90b~90dからの開始要求を受け付けると、ログ採取に関するシステムの動作状況に基づいて、トランザクション90b~90dの受け入れ可否を判断する。受け入れ判断基準としては、例えばログボリューム内の有効なログが占める割合を用いる。すなわち、有効範囲監視手段99によって有効なログとされたログがログボリューム中に占める割合が一定値以上である間は、トランザクション90b~90dの受け入れを拒絶する。

【0077】ログ採取手段96は、メタキャッシュ94内で変更されたメタデータの内容をログとして採取する。ログバッファ95は、ログ採取手段96が採取した情報を保持する。ログ書き込み手段97は、ログバッファ95が保持する情報を、適宜ログボリューム92に格納する。メタデータ書き込み手段98は、メタキャッシュ94内のメタデータをメタボリューム91内に格納する。有効範囲監視手段99は、メタデータ書き込み手段98による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応するログボリューム92内のログを、有効なログとして指定する。

【0078】このようなデータ処理装置によれば、トランザクション90b~90dから開始要求が出されると、トランザクション受け入れ制限手段90aが受け入れの可否を判断する。例えば、有効範囲監視手段99により有効であると指定されたログのログボリューム92内に示す割合が一定以上の場合には、それ以上ログが発生しないように、トランザクションの受け入れを拒絶する。その後、メタデータ書き込み手段98によるメタデータの書き込みが進み、有効なログの割合が減少したら、その時点でトランザクションの開始要求を許可する。

【0079】これにより、ログボリュームの空き容量の減少に伴うハングアップなどの障害の発生を防止することができる。次に、本発明の実施の形態を具体的に説明する。

【0080】図10は、本発明を適用するデータ処理装

置のハードウェア構成図である。データ処理システムは、CPU(Central Processing Unit)211を中心に構成されている。CPU211は、バス217を介して他の機器を制御するとともに、様々なデータ処理を行う。バス217には、メモリ212、入力機器インタフェース213、表示制御回路214、HDD(Hard Disk Drive)インタフェース215、及びネットワークインタフェース216が接続されている。

【0081】メモリ212は、CPU211が実行すべきプログラムや、プログラムの実行に必要な各種データを一時的に保持する。入力機器インタフェース213は、入力機器としてキーボード221とマウス222が接続されており、これらの入力機器からの入力内容をCPU211に伝える。

【0082】表示制御回路214は、表示装置223が接続されており、CPU211から送られてきた画像データを表示装置223で表示可能な画像情報に変換し、表示装置223の画面に表示させる。

【0083】HDDインタフェース215は、複数のHDD231~233が接続されており、CPU211から送られてきたデータをHDD231~233に格納するとともに、CPU211からの要求に応じてHDD231~233内のデータを読み取り、CPU211に転送する。

【0084】ネットワークインタフェース216は、LAN(Local Area Network)に接続されており、LANを介してデータ通信を行う。すなわち、CPU211から送られたデータをLANに接続された他のコンピュータに転送するとともに、他のコンピュータからLANを介して送られてきたデータをCPU211に転送する。

【0085】HDD231~233には、各種ファイルや、そのファイルを管理するためのメタデータ及びログが格納されている。このような構成のシステムにおいて、CPU211がHDD231~233に格納されたオペレーティングシステム用のプログラムを実行することにより、本発明のログ採取機能が実現される。

【0086】図11は、ファイルシステム上で動作するログ採取機能の構成図である。図のように、ファイルを管理するためのメタボリューム111~113も複数設けられている。メタボリューム111~113には、それぞれメタデータが格納されている。メタデータは、ファイルの格納場所等を管理するために必要な情報を有している。

【0087】ログボリューム120は、ログ122を格納するための二次記憶装置である。ログボリューム120には、ログ122の他にボリューム管理情報121が格納されている。

【0088】メタキャッシュ130は、メタデータを操作するためのメモリ上の領域である。メタキャッシュ130内には、操作対象となるメタデータ132とそのメ

10

20

30

40

50

タデータの割り当て管理情報131とが格納される。

【0089】ログキャッシュ140は、複数のログバッファ141～144を有している。ログバッファ141～144のサイズは均一ではなく、大きなサイズのものや小さなサイズのものがある。これらのログバッファ141～144には、メタキャッシュ130内で更新されたメタデータの複製がログとして格納される。

【0090】ログキャッシュ140とは別にログライトバッファ150が設けられている。ログライトバッファ150には、トランザクションの処理が終了した時点で、ログバッファ141～144内のログが転送される。

【0091】この例では、複数のトランザクション101～103が並列動作している。このトランザクション101～103は、ファイルシステムオペレーションを分割したものである。

【0092】実際にI/Oを行うのは2つのデーモンであり、それぞれをメタライトデーモン104、ログライトデーモン105と呼ぶ。ログライトデーモン105がログをログ専用の二次記憶装置であるログボリューム120に出力する。メタライトデーモン104は、ログボリューム120に対してログが出力されたことを確認した後、そのログに対応するメタデータをメタデータ専用の二次記憶装置であるメタボリューム111～113に出力する。

【0093】さらに、本発明のログ採取機能では、以下のような特徴を有している。第1の特徴は、各メタデータに対応するメタデータ管理情報として、メタボリュームを識別する情報が付加されていることである。これは、大規模ファイルシステムに対応するためのものである。すなわち、複数の二次記憶装置がメタボリューム111～113として定義される場合に、メタキャッシュ130上のメタデータ管理情報にそのメタデータが存在するボリュームの情報を持たせている。

【0094】図12は、メタデータ管理情報を示す図である。メタデータ管理情報には、「ボリューム番号」、「メタデータ番号」、及び「メタデータポイント」が登録されている。「ボリューム番号」には、対応するメタデータが存在するボリューム番号が登録されている。

「メタデータ番号」には、ボリューム毎におけるメタデータ管理番号が登録されている。すなわち、システムが認識するボリュームのデバイス番号とそのボリューム内の位置から算定される数値によってメタデータが管理され、メタデータ自体がそれらの値を管理情報として保持する。「メタデータポイント」には、メタデータの実体のある場所を指し示している。

【0095】このようなメタデータ管理情報を有するメタデータの内容が更新されると、更新後のメタデータの内容がログとしてログバッファに記録されるとともに、メタデータ管理情報の内容がログに記録される。

【0096】図13は、ログバッファの形式を示す図である。ログバッファには、「BEGINマーク」、「ボリューム番号」、「メタデータ番号」、「メタデータ実体」、及び「ENDマーク」の情報を含んでいる。

【0097】ファイルシステムを復元する際には、ログ内に記されているボリューム番号によって、そのログによって復元すべきメタデータの存在するメタボリュームが認識される。これにより、従来技術のようにファイルシステム復元時にメタデータ番号からその存在すべきボリュームを決定するよりも速度向上が望め、システムダウン後の大規模ファイルシステムにおいてもログ機構導入によるファイルシステム復元時間の短縮が有効に機能する。

【0098】第2の特徴は、メタキャッシュ130内で更新されたメタデータが、それぞれの管理構造にリンクポイントを持つことである。ログとしてログボリューム120に記録されたメタデータはメタライトリストに繋がれ、メタボリュームへ反映が完了した時点でこのメタライトリストから外される。さらに、ログとして記録されているログボリューム120内の位置情報をも、メタライトリスト内に持つ。このログボリューム内位置情報をリストを辿って検索することによって、システムダウン時に必要とされるログの範囲を特定することができる。そこで、ここから得られる有効範囲情報をログボリューム120の特定位置に設けたボリューム管理情報121内に記録する。有効範囲情報を記録し、ファイルシステム復元時にそこに含まれるログのみをリプレイすることにより、ログボリューム全体を検索する必要がなくなり、ログボリューム全体を読み込む必要のある、有効範囲情報を記録しない従来の方式よりもファイルシステムの復元時間をさらに短縮することができる。

【0099】ところで、ログ機構はシーケンシャル性を持ったディスクアクセスを行うことで、記録すべき内容をヘッドシークすることなしにディスクへ保存でき、ディスクアクセス時間の短縮を図っている。ところが、本手法を採用した場合、メタデータのメタボリューム反映時、ログボリュームへの書き出し時に、ログボリュームの特定位置に有効範囲情報を書き出すこととなる。これではシーク削減の意図が全く意味を成さない。

【0100】そのため、本実施の形態ではある程度のインターバルを空けて、有効範囲情報は書き出すように工夫した。変更がある度に、常に書き込まれるわけではないため有効範囲情報として保存されている情報には若干の誤差が含まれてしまう。ファイルシステム復元時にその誤差を吸収する必要がある。

【0101】図14は、有効範囲を説明する図である。図中において、「●」で示すのが、まだメタボリュームに反映が終了しておらず、ファイルシステム復元に必要なログを意味する。「○」で示すのは、メタボリュームに反映が完了したことによって、ファイルシステム復元

時には利用しなくても良いログである。

【0102】従来のファイルシステムにおいて、ログを用いたファイルシステム復元では、ログボリューム全体を検索し、ログに記録されたシーケンス番号から、最古のログを求め、たとえそれがメタボリュームに反映済みの、利用しなくても良いログであっても利用して、ログボリューム全体のログを用いていた。

【0103】本発明では、ある時点で有効範囲情報を書き出した時の有効範囲が示されている。その後、有効範囲を書き出さずに、メタボリュームへの反映が進み、また、別のトランザクションのログがログボリュームに書き出されたことによって、実際の有効範囲と有効範囲情報とはズレを生じている。この時点でシステムがダウンし、ログを用いてファイルシステムを復元する場合には、多少のムダが生じるが、有効範囲情報が示す先頭位置から、ログを利用する。利用すべきログの末端は有効範囲情報以降の一定範囲を検索しなければならないが、その検索は有効範囲情報を書き込むインターバルに依存し、範囲が限られている。

【0104】第3の特徴は、ログに対してトランザクション終了時にシーケンス番号を与え、その番号にはデータサイズが肥大化することを考慮に入れた上で、十分大きいデータ型を適用することである。データ型の大きさは、コンピュータ自身の使用可能年数の間使い続けても枯渇しない程度のサイズとする。例えば、システムの日時表記を4桁の十進数「1999」で表していた場合、西暦1万年まで使用されることは想定されていない。その場合、西暦9999年まで使用可能なデータ型とすれば、ログに対するシーケンス番号がオーバーフローして逆転することがないことを保証することができ、それにより、ログボリューム全体をゼロクリアして初期化する必要が生じない。具体的には、データ型を64ビット型とすれば、オーバーフローすることは現実にはありえない（4万年ほど耐えられるものと思われる）。ファイルシステム管理情報、各トランザクションのログ、有効範囲情報にこのシーケンス番号を含め、ログボリュームに記録する。このように、ログに与えるシーケンス番号のデータ型に十分大きいものを適用することにより、通常の運用時にトランザクションが動作する毎にインクリメントしても、オーバーフローは現実には起こり得ない。

【0105】さらに、スーパブロックと呼ばれるファイルシステム全体の管理情報にこのシーケンス番号を含め、正常なアンマウント処理時及びファイルシステム復元時にスーパブロックに含まれるシーケンス番号を正しく設定し、次のマウント時にそこから得られる値を用いる。これにより、シーケンス番号は必ず昇順となることが保証され、ファイルシステム復元時に新しいログを古いものと取り違えないことが保証される。

【0106】シーケンス番号の順序性を保証することによって、ファイルシステム復元時にログボリュームをゼ

ロクリアしなくとも、常に正しいログを利用することが可能となり、ファイルシステム復元時にログボリューム全体をゼロクリアするのに比べ、大幅な時間短縮が可能となる。

【0107】以上の理由により、ログの最大のメリットであるファイルシステム復元の時間短縮がより一層有効に機能することが可能となる。第4の特徴は、トランザクションによって更新されたメタデータが、そのメタデータの属性に応じ、再度同一のトランザクションにおいて更新される可能性のあるメタデータであった場合には、更新の時点ではログバッファにはコピーせず、リスト構造（トランスリスト）によって管理することである。

【0108】そして、同一トランザクション内で複数回更新されるメタデータとして、領域割当て管理情報をリスト構造で管理し、トランザクション進行中にはその中途段階のログは採取しない。トランザクション終了時にリスト構造を辿って、トランザクションの更新の最終状態のみを一括してログとすることによって、ログの縮小が図られ、それに伴いファイルシステム復元時間の短縮が可能となる。

【0109】具体的には、メタキャッシュ130内のメタデータを管理する構造にトランスリストへのリンクポイントを持たせることによって実現している。トランザクションによって更新されたメタデータは、ただ一回しか更新されないことが分かっているメタデータである場合には、その時点でログバッファへコピーされるが、以降もトランザクション進行中に更新される可能性がある場合には、このリンクポイントを用いてリスト構造に繋がれる。メタデータの更新可能性の有無は、トランザクションの種別で判断する。例えば、データ領域の空きなどを管理するためのトランザクションでは、何度もメタデータが更新される。

【0110】そして、トランザクションが終了する時にこのトランスリストを辿り、メタデータの最終形態をログバッファへコピーする。すると、結局はトランザクションが同一メタデータを複数回更新しても、最終形態の一回だけのログ採取で済まされる。

【0111】図15は、ログ採取処理のフローチャートである。この処理は、オペレーティングシステムを実行するCPUが行う処理である。以下、CPUがオペレーティングシステムを実行することにより実現する機能を、単に「システム」ということとする。

【S1】トランザクションの開始宣言を行う。

【S2】ログ採取要求を行う。

【S3】対象メタデータの更新可能性の有無を判断する。更新可能性があればステップS5に進み、そうでなければステップS4に進む。

【S4】ログバッファへメタデータをコピーし、ステップS2に進む。

10

20

30

40

50

【S5】対象メタデータはトランスリストに繋がれているか否かを判断する。トランスリストに繋がれていればステップS7に進み、そうでなければステップS6に進む。

【S6】メタデータ毎にトランスリストに繋ぐ。

【S7】トランザクションの処理が終了するか否かを判断する。終了するのであればステップS8に進み、そうでなければステップS2に進む。

【S8】トランザクション終了宣言を行う。

【S9】トランスリストを辿り、繋がれている最終形態のメタデータをそれぞれログバッファへコピーする。

【0112】このようにして、トランザクションの更新の最終状態のみを一括してログとして保存することができる。第5の特徴は、メタボリュームの割当て管理情報は獲得用として通常の割当て管理情報の複製を新たに設けたことである。ここで、割り当て管理情報として、ビットマップを例に挙げて説明する。

【0113】図16は、メタボリュームの割り当て管理状況を示す図である。割り当て管理情報131には、使用されているメタデータを管理するためのビットマップ131aが用意されている。ビットマップ131aは複数のブロックに分けられている。そして各ブロックのビットマップの各ビットが「0」か「1」かによって、対応するメタデータが空いているか否かが示される。そして、ブロックに分けられたビットマップの1つの複製が作られ、獲得用ビットマップ131bとされる。

【0114】獲得時には、通常のビットマップ操作と同様に未割り当て状態の領域の検索が行われる。この時、検索対象として用いるのは獲得用ビットマップ131bである。獲得用ビットマップ131b内から未割り当て状態の領域（ビットが立っていない）が見つかった時には、その獲得要求には見つかったビット番号を返す。そして、獲得用ビットマップ131b及び、その複製元となったビットマップの対応ビットを立てる。一方、獲得用ビットマップ131b内から未割り当て状態の領域が見つからなかった時には、別のブロックのビットマップの複製を生成し、新たな獲得用ビットマップ131cとする。

【0115】図17は、ビットマップによるメタデータ獲得処理を示すフローチャートである。この処理は、システムが行う処理である。

【S11】獲得要求を発行する。

【S12】獲得用ビットマップに空きがあるか否かを判断する。空きがあればステップS20に進み、そうでなければステップS13に進む。

【S13】メモリ（メタキャッシュ130）上のビットマップに「FreeDirty」フラグが立てられていないビットマップが存在するか否かを判断する。存在すればステップS14に進み、存在しなければステップS16に進む。ここで「FreeDirty」フラグとは、一回以上の解放

処理が対象ビットマップに対してなされたことを意味する。

【S14】「FreeDirty」フラグが立てられていないビットマップの中で、空きのあるものがあるか否かを判断する。そのようなビットマップがあればステップS15に進み、そうでなければステップS16に進む。

【S15】「FreeDirty」フラグが立てられておらず、空きのあるビットマップの複製を、獲得用ビットマップに作成する。その後、ステップS20に進む。

【S16】メタボリューム111～113上のビットマップに空きがあるか否かを判断する。空きのあるビットマップがあればステップS17に進み、そうでなければステップS18に進む。

【S17】メタボリューム111～113上の空きのビットマップをメモリ（メタキャッシュ130）上に読み込み、ステップS15に進む。

【S18】メモリ（メタキャッシュ130）上のビットマップに「FreeDirty」フラグが立てられたビットマップが存在するか否かを判断する。存在すればステップS19に進み、存在しなければ、獲得不可能と判断し処理を終了する。

【S19】「FreeDirty」フラグが立てられたビットマップをメタボリュームに反映し、「clean」状態にする。ここで「clean」状態とは、獲得、解放の処理が全く行われていない状態を示す。その後、ステップS13に進む。

【S20】獲得用ビットマップのビットを立てる。

【S21】複製元ビットマップの同一ビットを立てる。

【S22】複製元ビットマップに「AllocDirty」フラグを立てる。ここで、「AllocDirty」フラグとは、対象ビットマップから一回以上の獲得処理によって更新がなされた場合に、そのビットマップの管理構造体内のフラグに立てる値である。「AllocDirty」フラグや「FreeDirty」フラグが立ったビットマップはログ採取対象である。メタボリュームに反映された時にこれらのフラグは落とされ、「clean」状態となる。

【0116】このようにして、空きのビットマップを獲得できる。解放時にも、通常のビットマップ操作と同様に、要求された領域が対応するビットの算出がまず行われるが、対応するビットを落とす操作は、対象のビットマップに対してのみ行い、仮にその対象ビットマップの複製が獲得用ビットマップとして存在する場合にも、その獲得用ビットマップに対しては行わない。

【0117】図18は、解放処理のフローチャートである。この処理は、システムが行う。

【S31】解放要求を出す。

【S32】対象ビットマップがメモリ（メタキャッシュ130）上にあるか否かを判断する。対象ビットマップがあればステップS34に進み、そうでなければステップS33に進む。

【S33】メタボリューム111~113上の対象ビットマップをメモリ（メタキャッシュ130）上に読み込む。

【S34】対象ビットマップの対応するビットを落とす。

【S35】対象ビットマップに「FreeDirty」フラグを立てる。

【0118】この一見複雑に見える作業によって、解放した領域を即座に別の用途に利用してしまうことを回避することが可能となり、システムダウン時に中途までしか終了していなかった解放トランザクションが解放したはずである領域は、解放される直前の状態のまま保全されることが保証できる。

【0119】例えば、Aというトランザクションが解放処理、Bというトランザクションが獲得処理を行う場合を考える。ここで、トランザクションAの解放処理はトランザクションBの獲得処理よりも先に行われ、かつ、トランザクションAはトランザクションBよりも後に終わる場合を例に挙げる。

【0120】図19は、トランザクションの処理の獲得と終了の状況を示す図である。この図において、各トランザクションは「BEGIN」で始まり、「END」で終わることを意味し、トランザクションの「○」が解放処理を、「●」が獲得処理を意味する。

【0121】上図のように、トランザクションAの解放処理がトランザクションBの獲得処理よりも先に行われ、かつ、トランザクションBのほうがトランザクションAよりも先に終了した場合に従来のログ採取方式では問題が生ずることは既に述べた。

【0122】本発明が提案する手法を用いれば、トランザクションBが獲得する領域がトランザクションAが解放した領域となることは基本的にはなく、領域枯渇状態に近く、どうしてもこの領域を獲得せねばならない時には、一旦、該当ビットマップをメタボリュームに反映した後、獲得処理が行われる。これにより、ファイルシステム復元時に必要とされる、メタボリュームに未反映なメタデータを対象とするログを用いた復元では、管理情報上、二重獲得と見えるようなログは残り得ない。

【0123】また、トランザクションBのログにはトランザクションAの解放処理が記録されないため、トランザクションBのログよりも後に復元に用いるトランザクションAのログによって、トランザクションBが獲得した領域を変更されることもありえない。

【0124】第6の特徴は、前述のようにログ機構の導入にあたり、獲得・解放操作のログとして、その獲得・解放した対象の情報（このビットを立てた・落した）のみを記録することである。これによって、複数の獲得・解放要求に対して、要求の発生後、トランザクションの終了時点までをシリアルライズすることなくログ管理でき、かつログ採取量は減少し、複数トランザクション動

作による変更をメタボリュームへ反映する回数をも減少させることができる。

【0125】第7の特徴は、ログキャッシュ140を複数のログバッファで管理し、この時、いくつかの異なるサイズとすることである。ログキャッシュ140を分割して複数のログバッファとして管理することによって、トランザクションの独立性を確立することが容易となるとともに、有限なメモリ空間を有効に活用することが可能となる。

【0126】まず、トランザクションがメタデータを更新するより以前に、自分がメタデータを更新する旨、システムに宣言する。この時、トランザクションの種別より予測されるログ採取情報に応じて、最適なサイズのログバッファがシステムより与えられ、トランザクション進行に伴い蓄積されるログはここへ溜められることとなる。上記第6の特徴で説明した手法に加え、トランザクション毎に異なるログバッファを用いることによって、複数のトランザクション更新の混じらない、純粋なログとして残すことが可能である。

【0127】第8の特徴は、上記第7の特徴で説明した手法に加え、トランザクションの開始時に予測したログ採取量よりも多くのログを採取しなければならなかった場合に、与えられたログバッファから、さらに大きいログバッファへ移行することである。システムの状況に応じて、トランザクションによって残されるログのサイズは変動し、その差異を複雑な処理を必要とせず、かつ、有限なメモリ空間を有効に活用して吸収することが可能である。

【0128】第9の特徴は、ログバッファに入りきらない場合には、ログボリュームへ中間ログを書き出すとともに、中間ログを書き出すことによる矛盾の発生を防止する機能を備えたことである。

【0129】ログボリュームに対するI/O発行を減少させるために、ログライトバッファと呼ぶ二次キャッシュ領域を設けている。終了したトランザクションのログは、ログバッファからこのログライトバッファへ移され、適時にログライトデーモン105によってログボリュームへ書き出される。この適時とは、負荷が低い場合には一定の周期で良いが、トランザクションによるメタデータの更新量が多く、最大のログバッファでも収まらない場合などには強制的なI/O発行が必要とされる。このような場合において、中途の段階でのログの出力では、まだ更新されていない（かもしれない）メタデータを含めた書き出しをファイルシステムの整合性を保つためには必要である。

【0130】このように有限なメモリ空間を有効活用することを考慮した上で、ログバッファ内に収まりきらないログをトランザクション途中で出力する時に、残されるログによって復元されるファイルシステムの整合性を正当なものとするための手法を提案している。

【0131】ファイルシステムに対する負荷がそれほど高くない場合には、ログのログボリュームに対するI/Oを極力少なくするために事前に与えられた周期に応じて定期的にデーモンによって行われる。しかし、ログバッファが枯渇し、以降のトランザクション進行で溜められるログが収まらないと判断された時、部分的なログとして、この時点のログをログボリュームへ出力する。この時、まだ更新されていない情報をもログに出力する。例えばファイルを追加ライトするトランザクションの場合は、部分的なログを出力する時点でのファイルサイズや時刻情報を合わせてログに記録する。これによって、部分的であるはずのこのログを、ファイルシステム復元時には完結した1つのトランザクションのログと同等に扱うことが可能となり、ファイルシステム復元時に複雑な考慮の必要なしに、望ましい状態にファイルシステムを復元することが可能となる。

【0132】図20は、ログバッファへのログの格納状況を示す図である。この例では、2種類のサイズのログバッファ141～146が設けられている。ログバッファ141～145は通常のサイズであり、ログバッファ146が大きなサイズである。

【0133】また、ログバッファ141～146を管理するためのログバッファ管理テーブル148、149が設けられている。ログバッファ管理テーブル148は、ログバッファ141～146に対応するフラグビットを有している。そして、使用されているログバッファに対応するフラグビットの値が「1」に設定され、未使用のログバッファに対応するフラグビットの値は「0」に設定されている。同様に、大きいサイズのログバッファ146に対応するログバッファ管理テーブル149もフラグビットを有しており、そのフラグビットの値によって、ログバッファ146が使用されているか否かを管理している。

【0134】ログキャッシュ140内の各ログバッファ141～145のうち、ログバッファ143とログバッファ145とが現在利用中であることを意味するために、「●▲」などの記号を用いた。ここで、ログバッファ145には多くのログが溜まっており、既に満杯の状態となっている。このように通常のサイズのログバッファ141～145では容量が足りなくなった際には、そのログを大きいサイズのログバッファ146に移動する。そして、トランザクションが継続している間は、更新されたメタデータの内容を、随時ログバッファに格納していく。ここで、トランザクションが終了したら、そのトランザクションに対応するログバッファの内容をログライトバッファ150へ転送する。

【0135】ログライトバッファ150には、複数のバッファ151、152が設けられており、それらのバッファ151、152にログが書き込まれる。ログライトバッファ150内のログは、ログライトデーモン105

によって随時ログボリュームに書き込まれる。

【0136】図21は、ログ採取手順を示すフローチャートである。これはシステムが行う処理である。

【S41】BEGIN宣言を行う。

【S42】ログバッファの予約をする。

【S43】ログ採取要求を行う。

【S44】現在のログバッファに今回のログが収まるか否かを判断する。収まるのであればステップS51に進み、収まらないのであればステップS45に進む。

【S45】現在のログバッファより大きいログバッファが存在するか否かを判断する。存在すればステップS46に進み、そうでなければステップS47に進む。

【S46】現在のバッファから大きいバッファへ内容をコピーし、ステップS44に進む。

【S47】トランザクションに与えられたパラメタをログ採取する。

【S48】現時点のファイル状態を正しく表す管理情報をログ採取する。

【S49】現在の中段階のログをログライトデーモン105に書き出させる。

【S50】現在のログバッファをクリアする。

【S51】ログを採取する。

【S52】ログ採取要求を終了する。

【S53】END宣言あるか否かを判断する。END宣言があればステップS54に進み、そうでなければステップS43に進む。

【S54】END宣言を行い、処理を終了する。

【0137】第10の特徴は、トランザクションが完了する前に、システムダウンが発生した場合に備え、分割して出力するログにはトランザクションに与えられたパラメタを含め、ファイルシステム復元時にそのパラメタを元に、再度トランザクションを実行することである。

【0138】さらに、この部分的なログにトランザクションに与えられたパラメタを記録し、ファイルシステム復元時にそれを用いて、通常のログだけでは途中で終わった状態までしか復元できないファイルを、トランザクションが終了した状態にまでファイルシステムに反映することが可能となる。

【0139】図22は、ファイルシステム復元処理を示すフローチャートである。これはシステムが行う処理である。

【S61】システムダウン等が発生する。

【S62】ファイルシステムの異常を検知する。

【0140】以降、ファイルシステム復元処理を行う。

【S63】ログ開始マークを検出する。

【S64】ログ開始マークに対応するログ終了マークが存在するか否かを判断する。ログ終了マークが存在すればステップS65に進み、ログ終了マークが存在しなければステップS66に進む。

【S65】開始マークから終了マークまでのログを用い

て復元処理を実行する。その後、ステップS63に進む。

【S66】対象トランザクションが以前のログだけで完結するか否かを判断する。完結するのであれば復元処理を終了し、完結しないのであればステップS67に進む。

【S67】ログに記録されたパラメタを読み込む。

【S68】トランザクションとして行いたかった処理を理解する。

【S69】ファイルに対して直接処理を再実行する。その後、復元処理を終了する。

【0141】これにより、オペレーションのセマンティクスを保証したファイルシステムの復元が可能となり、従来技術では完全に元に戻す、または完全に再実行することが不可能であったトランザクションを、完全に再実行することによってファイルの整合性をも回復することが可能となる。

【0142】第11の特徴は、メタデータを更新するトランザクションが更新処理を行う前に、その旨をシステムに対し宣言することである。さらに、システムは宣言を受け入れるか否かを判断する機構を持つことである。ここでは、以下の条件の場合には新規のトランザクションの受け入れを拒否し、拒否されたトランザクションは再度認可が下りるまで待たなければならない。

- ・メタキャッシュ130がフルに近い状態にある場合。
- ・トランザクションの多重度が、システムで規定された値を既に越えていた場合。
- ・ログボリュームに残されたログの大部分が有効なログ状態である場合。

【0143】これらの場合に、ファイルシステムが新規トランザクションの受け入れを拒否し、トランザクションの多重度を宣言することによって、結果としてダーティなメタデータの数制限することが可能となり、メタキャッシュ130の空き領域枯渇などを要因とするハングアップ状態に陥ることを回避することが可能となる。特に、分割ログとしてログ採取しなければならないトランザクションが動作中に、新規トランザクションの開始を拒否することは、システムの状態を正常に維持する上で効果が高い。

【0144】第12の特徴は、新規トランザクションの受け入れ拒否の機構をログボリューム内の有効ログの割合に応じて適用することにより、単一の、多数のメタデータを更新するトランザクションのみならず複数のトランザクションが更新したメタデータによって空き領域の減少に伴うハングアップ状態をも回避することである。

【0145】以下に、第11の特徴と第12の特徴とを含むトランザクションの受け入れ処理について説明する。図23は、新規トランザクションの受け入れ許可判定処理を示すフローチャートである。

【S71】トランザクションを行うプロセス（以下、単

にトランザクションという）が、トランザクション処理を開始する。

【S72】「BEGIN」宣言を行う。この際、オペレーティングシステム（以下、単に「システム」という）に対して問い合わせを行う。

【S73】トランザクションは、システムからの回答待ち状態となる。システムより「OK」の回答を受け取ったらステップS82に進む。

【S74】トランザクションからの問い合わせを受けたシステムが、パラメタの評価を開始する。

【S75】システムは、既に多重度が規定値より上であるか否かを判断する。多重度が規定値より上であれば「NG」としてステップS81に進み、そうでなければ「OK」としてステップS76に進む。

【S76】システムは、現在サブトランザクションが動作中であるか否かを判断する。サブトランザクションが動作中であれば「NG」としてステップS81に進み、そうでなければ「OK」としてステップS77に進む。ここでサブトランザクションとは、ログを複数に分割して格納する場合に、1つのログバッファにログを格納できるような処理単位に分割されたトランザクションである。

【S77】システムは、メタキャッシュ130に十分な空きがあるか否かを判断する。十分な空きがなければ「NG」としてステップS79に進み、十分な空きがあれば「OK」としてステップS78に進む。

【S78】システムは、ログボリュームに上書き可能領域が少ないか否かを判断する。上書き可能領域が十分になれば「NG」としてステップS79に進み、十分な上書き可能領域があれば「OK」としてトランザクションに対して「OK」の回答を返す。

【S79】システムは、ログライトデーモン105を起動する。

【S80】システムは、メタライトデーモン104を起動する。

【S81】システムは、NG条件が解消されるまでスリープ状態で待つ。NG条件が解消されたら、ステップS75に進む。

【S82】トランザクションは、システムからの「OK」の回答を受け取ったら、多重インクリメント処理を行う。

【S83】トランザクションは、「BEGIN」処理を終了する。

【S84】トランザクションは、処理に応じて、メタデータをキャッシュに読み込み、その度に空き量をデクリメントする。分割ログ化するなら他トランザクションの開始を拒否するようにシステムに依頼する。

【S85】トランザクションは、「END宣言」を行う。

【S86】トランザクションは、多重度をデクリメントする。

【S87】トランザクションは、必要に応じてログ採取を繰り返した後、自己のトランザクション処理を終了する。

【0146】次に、本発明を適用したシステムの具体的な処理内容について説明する。本発明の全ての特徴を備えたシステムにおけるログ採取手順は以下になる。ファイルシステムオペレーションを分割したトランザクションはメタデータを更新するより前に、自分がこれからメタデータを更新する旨を宣言する(BEGIN)。BEGIN時にトランザクションに対し、独立したログバッファが割当てられる。この時、既にトランザクションの並列動作数が非常に多かったりメタキャッシュ130域の利用率が高い場合には、システムによってBEGIN宣言が拒否される。BEGIN宣言を拒否されたトランザクションはその許可理由が解消されるまで、待ち合わせしなければならない。

【0147】更新が完了したメタデータは、BEGIN時に割り当てられたログバッファへコピーされる。それと同時にメタデータ種類に応じたリストへ繋がれる。更新すべき全てのメタデータを更新し終わったトランザクションは、そこで完了を宣言する(END)。END時には、これまで溜めたログバッファの内容をログ専用の二次キャッシュであるログライトバッファへ移動し、さらに、まだログバッファへコピーされていなかったメタデータをログライトバッファへコピーする。

【0148】また、END時にメタデータの種類に応じて繋がれていたリストから、そのトランザクションが更新した全てのメタデータを「ログ待ちリスト」へ繋ぎ替える。

【0149】以上で非同期要求のトランザクションは終了することができる。トランザクションが同期要求であった場合には、ログを書き出すまで待ち合わせなければならない。

【0150】ログの書き出しは独立したデーモン(ログライトデーモン105)が行う。このデーモンの起動契機は、同期要求トランザクションによる起動要求(wakeup)、メタキャッシュ130の空き状態監視機構による起動要求(wakeup)、及びタイマである。

【0151】起動したログライトデーモン105は複数のトランザクションのログがまとめられたログライトバッファを1つのI/Oとして発行する。そして、ログ待ちリストから「書き出しリスト」へメタデータを移動する。

【0152】メタボリュームへI/Oを発行するのは、メタライトデーモン104の役割である。メタライトデーモン104は書き出しリストに繋がれたメタデータを順に非同期ライトによってメタボリュームに反映する。メタライトデーモン104の起動契機は、ログバッファの空きが少なくなった時、ログボリュームの空きが少なくなった時、及びタイマである。

【0153】以上で、メタデータを更新するトランザクションの開始から、更新されたメタデータがメタボリュームに反映されるまでの大まかな流れである。以降に、ログの採取とそのディスク反映手法、そして、メタデータのディスク反映の処理について詳細に説明する。

(1) ログの構造

(1. 1) ログボリューム

ログボリュームに格納される情報は、以下のような情報である。

【0154】ログボリュームにはスーパーブロック、ボリューム管理情報が先頭にある。その次にログの有効範囲を示す構造体を記録する。構造体は以下のメンバを持つ。

- ・有効ログ先頭のログボリューム内オフセット
- ・有効ログ先頭のログシーケンス番号
- ・有効ログ末尾のログボリューム内オフセット
- ・有効ログ末尾のログシーケンス番号

ただし注意しなければならないのは、ここで先頭・末尾と言っているのは、真の値ではない可能性があることをリプレイ時に考慮しなければならない点である。なぜなら、ログはボリュームをシーケンシャルアクセスすることによってシーク時間の短縮を計っているが、このようにボリュームの一部へ毎回アクセスしたのでは、そのシーケンシャル性が損なわれてしまう。そのために、ログの書き出し毎にはこの有効範囲の書き出しは行わず、数回に一回、書き出している。

【0155】これにより、上記構造体に収められたオフセットは正確ではないためにリプレイ時に検索が必要ではある。しかしながら、全体を検索するよりも大幅に検索量が減るため、利点は保持できるものとする。

【0156】上記以外は、全て、メタデータの更新履歴によって構成される。

(1. 2) ログブロックの基本構造

ログボリューム120内に残されたログ(メタデータの更新情報)は基本的にはトランザクション単位である。これをログブロックと呼ぶ。しかし、キャッシュ域フルなどによって、1つのトランザクションを一度にまとめることができない場合は複数の分割ログに分ける。この分割ログをサブトランザクションログと呼ぶ。

【0157】サブトランザクションログをリプレイすることによって、ファイルシステムの整合性が保てるよう、その内容は工夫されている。しかしながら、トランザクションの途中までのサブトランザクションをリプレイしたのでは、トランザクション全体が終わっていないことから、ファイルシステムとしての整合性が保てても、ファイルの中身は異常であるという状態に陥ってしまうことが考えられる。

【0158】この状態を回避するために、そのサブトランザクションに別れたトランザクションがどのようなオペレーションであったかをログに採取する(オペレーシ

ョンログ)。ログリプレイ時には、サブトランザクションをリプレイした後、オペレーションログをリプレイヤが再実行することによって、トランザクション全体が終わった状態とすることが可能である。

【0159】サブトランザクション化する可能性のあるトランザクションは全体終了時に「終わったこと」をログに記録し（最終ENDマーク）、リプレイヤはその有無を調べてオペレーションログの実行を判断する。

(1. 3) ログの採取形式

メタデータの更新情報は該当するメタデータの管理構造単位で採取する。また、メタボリュームの管理構造であるビットマップはその更新部分だけのログ採取とする、スーパーブロックについては特殊であり、データ空き容量についてのみログ採取を行う。

(1. 3. 1) inode、Vデータ、空き管理情報
ファイルの管理情報であるiノード、Vデータと呼ぶディレクトリやシンボリックリンクデータや空き管理情報のメタデータ本体は、それらのI/O単位（ブロック単位）でのログ採取を行う。

【0160】これは、ファイルシステム整合性チェック処理であるfsckによるログリプレイ時の処理を簡略化することを意識している。変更された部分だけをログ採取した場合、ログリプレイ時には、該当ブロックの算定→読み込み→変更部分の更新→再度書き込み、とステップが増えてしまう。しかし、ブロック全体のログ採取によって、リプレイ時にはメタボリュームにログ情報を上書きするだけでよい。

【0161】iノード本体やディレクトリブロックなどのVデータは、更新中はファイルのロックで保護されている。しかしながら、空き管理情報についてはファイルのロックでは保護されない。そのため、トランザクション途中で他トランザクションによって更新され、そのトランザクションが追い越して先に終わってしまうと、ログには古い情報が後に残されてしまい、リプレイするとファイルシステムに不整合が生じてしまう。

【0162】そのため、空き管理部については、ここでは更新するトランザクションが並行動作しないことを保証することによって実質的な排他制御を行い、他のメタデータと同じ採取形式とした。

(1. 3. 2) ビットマップ

メタデータの割り当て状況を管理するビットマップも空き管理部と同様にファイルのロックで保護されていない。そのため、ビットマップ全体のログ採取を行うと、そのログには複数のトランザクションによる更新が含まれてしまう可能性がある。特に、トランザクションの追い越しが発生すると、新しいはずのログによって、ビットマップの情報が古く書き戻されてしまうことが考えられる。

【0163】そこで、ビットマップのログ採取は更新部分だけとする。更新部分とは、あるビットが0になっ

た、あるいは1になったと記録するだけである。それにより、トランザクションが並行動作してもそれぞれのトランザクションが更新した内容のみがログに残されるため、メタデータの後退は起こり得ない。

【0164】注意すべきは、ある領域を解放（「1→0」）した後、他トランザクションがそこを獲得（「0→1」）した場合である。トランザクションの追い越しが発生すると、ログリプレイによって獲得した領域を解放してしまう。

【0165】これについては、アロケート用ビットマップを1つ定め、複製を用いて操作を行うことによって回避する。

(1. 4) ログの記録構造

(1. 4. 1) 一般ログ

有効状態であっても、ある程度の複数スレッドが同時に進行することを許す。これは性能要件より必須である。しかし、ログボリューム内のログは前述の通り、トランザクション毎に保存する。1つのトランザクション結果をログに記録する際、ログは以下のパーツで構成する。

1) BEGINマーク

2) ヘッダ

3) 更新内容

(2) + 3) の繰り返し

4) ENDマーク

これを以下、ログブロックと呼ぶ。ログブロックはログボリュームの物理ブロック境界から始まるが、その終わりは物理ブロック境界であるとは限らない。

1) BEGINマーク

トランザクションの開始時に作成される情報であり、以下の内容を含む。

・マジックワード

・トランザクションタイプ

・ログシーケンス番号

・ログブロックサイズ

マジックワード：1つのトランザクションが始まったことを示すマジックワードを埋め込む。

【0166】トランザクションタイプ：BEGINマークにトランザクションのタイプ（種類）を埋め込むことによって、その後の更新情報に含まれる内容の概要を事前に知ることが可能。

【0167】ログシーケンス番号：ログに記録されたトランザクション毎にインクリメントされる数値。このカウンタが最小のログブロックがそのログボリューム内で最も古いトランザクションであることを示す。カウンタは64ビット型とし、オーバーフローすることは現実にはありえない（4万年ほど耐えられるものと思われる）。ログリプレイ時に、ログボリュームをゼロクリアすることにより、再度0から開始することも考えられるが、リプレイ時間の短縮を考慮し、利用したログの最後のシーケンス番号より昇順に採番することによりゼロク

リアを回避する。

【0168】ログブロックサイズ：ログブロックのENDマークまでのサイズを記録する。BEGINマークの先頭からENDマークの先頭までのサイズである。リプレイ時にはBEGINマークの先頭からこのサイズだけ移動し、ENDマークの情報を参照して、ログの有効性を判定する。

2) ヘッダ

更新されたメタデータについて、その保管位置を特定するための情報であり、以下のメンバによって構成される。

- ・メタデータタイプ
- ・メタボリューム番号
- ・ボリュームローカルメタデータ番号

メタデータタイプ：更新情報の後方にある又データ更新内容が何のメタデータであるのかを特定するために用いられ、リプレイヤはここから更新内容のサイズを判断する。

【0169】メタボリューム番号：メタデータ管理で用いているメタボリューム毎の番号を記録する。リプレイ時には、この番号から書き戻すメタボリュームを決定する。ボリュームローカルメタデータ番号：メタデータ管理で用いている、メタボリューム毎、メタデータ毎に0から始まる値を記録する。リプレイ時には、この番号からメタボリューム内のブロック位置に変換し、更新内容を書き戻す。これはブロック位置変換を削減することによるログ採取の高速化が狙いである。対象がビットマップである場合には、変更されたビットが該当するメタデータ本体のメタデータ番号を記載する（ビットマップ番号ではない）。

3) 更新内容

メタデータ本体の場合には、更新内容が含まれるブロック（それぞれのメタデータの管理単位）である。例えば、ディレクトリブロックが更新された場合には1024バイトのディレクトリブロック全体である。

【0170】ビットマップの場合には、全体ではなく、ここには「0→1」または「1→0」という情報だけ記録する。リプレイ時にはヘッダから該当するビットマップを判定し、それを読み込み、ここの内容から該当ビットを変更して書き戻す。

4) ENDマーク

BEGINマークに対応するマジックワード、ログシーケンス番号、ログブロックサイズを記録する。

【0171】ENDマークがマジックワードとログ番号だけでは、メタデータの内容によってはそれが偽りのENDマークとして見えてしまい、リプレイの時に処理を誤る可能性がある。これを回避するために、ENDマークは固有形式（メタデータを誤認することのない形式）としなければならない。

【0172】BEGINマークを固有形式としないの

は、BEGINマークだけ書き出された時点で、システムダウンした場合を考慮し、いずれにせよENDマークの識別ができなければならないからである。

【0173】固有形式とするために、固有の数値を64バイト分続ける。これにより、全てのメタデータがENDマークに化けることはなくなる。これに続けて、マジックワード、カウンタ（ログ番号）、ログブロックサイズを記録する。以降、ブロック境界まで、上記数値を埋める。BEGINマークに対応したENDマークを見つけることができないログ情報はリプレイしてはならない（ログ書き出し途中でシステムが異常終了したことを意味する）。

(1. 4. 2) 巨大トランザクションのログ

トランザクションが多く、メタデータを更新する場合には、ログバッファサイズ、ログボリュームサイズが有限であることから、複数のサブトランザクションログに分割する。サブトランザクションログはそれだけのリプレイによって、ファイルシステムの整合性は保たれる。しかし、トランザクション全体のサブトランザクションログをリプレイしなければ、ファイルの整合性が保てない。そのため、サブトランザクション途中でのシステムダウンを考慮し、オペレーションログ内部に含む。

1) BEGINマーク

2) オペレーションログ

3) ヘッダ

4) 更新内容

(3) + 4) の繰り返し

5) ENDマーク

(1) ~ 5) の繰り返し

30 (5') 最終ENDマーク)

1) BEGINマーク

一般ログのBEGINマークと同じ。ただし、サブトランザクションログとなりうるトランザクションは、書き込みや削除など、データ領域を触るものや、領域管理に関わるものに限定される。

【0174】これらのトランザクションがBEGINマーク内で設定されていたら、そのログはサブトランザクション化している可能性があり、必ずオペレーションログが続いている（たとえ単一のサブトランザクションログで構成されていても）。

40

2) オペレーションログ

サブトランザクション化したトランザクション（関数）に与えられた引数をオペレーションログとして保存する。巨大トランザクションを対象としたBEGINマークの直後には必ずオペレーションログを配置する。オペレーションログは、最初にトランザクションへ渡された情報（このファイルをこれだけのサイズトランケートするか、このファイルをこれだけアペンドライトするなど）である。

50

【0175】リプレイ時には、リプレイヤがファイルシ

システムを直接操作して、ここで残されたオペレーションログを実行しなければならない。オペレーションログは以下のメンバで構成される。

- ・パラメタ1
- ・パラメタ2
- ...

リプレイヤはBEGINマーク内のトランザクションタイプを見ることによって、オペレーションログ部分に並ぶパラメタ群のサイズ及び内容を知ることができる。

3) ヘッダ

一般ログのヘッダと同じ。

4) 更新内容

一般ログの更新内容と同じ。

5) 5') ENDマーク、最終ENDマーク

一般ログのENDマークと同じ意味合いを持ち、BEGINマークに対応するENDマークあるいは最終ENDマークが見つからない場合には、BEGINマーク以降の更新内容をリプレイしてはならない。

【0176】巨大トランザクションがサブトランザクションに分割されず、単一のログブロックのみの場合には、ENDマーク部分には最終ENDマークが書き出される。複数のサブトランザクションログに分かれている場合には、最後のログブロックだけ、ENDマークが最終ENDマークに化ける（最後以外のログブロックには通常のENDマークが書かれている）。

【0177】ENDマークは一般ログのENDマークと同一であり、マジックワード、カウンタ（ログ番号）、ログブロックサイズが記録されており、ブロック境界まで固有数値が埋まる。

【0178】最終ENDマークとENDマークの差は、マジックワードのみである。最終ENDマークがトランザクション全体の終了を表すことから、とても巨大なトランザクションが多くのサブトランザクションログに分割されている場合には、全ての処理が完了する時にこの最終ENDマークを出力する。すなわち、巨大トランザクションとして定義されるトランザクションについて、通常のENDマークで終わるログブロックがいくつか見つかったのに、最終ENDマークで終わるログブロックが存在しない場合には、それは巨大トランザクションの途中でシステムダウンが発生したことを意味し、ログリプレイヤがオペレーションログのリプレイを行わなければならないことを意味する。

(2) ログの採取

(2.1) ログバッファの管理

ログをトランザクション単位にログボリューム120へ出力するためには、メタデータの更新情報を一度、メモリ内に溜める必要がある。これをログバッファと呼ぶ。ログバッファはマウント時にまとめて用意し、ファイルシステム利用中にメモリの追加獲得は行わない。

【0179】トランザクション毎のログ採取とし、ま

た、トランザクション動作中に他のトランザクションが並行動作することを狙い、ファイルシステム毎にログバッファを複数持つ。その数は並行動作を許すトランザクション数によって決められる。

【0180】並行動作数を限定することはログ機能追加による性能劣化要因の1つとなるが、

- ・ログリプレイ時の処理を単純化
- ・巨大なログバッファを分割して使うことによる複雑さの回避

10 などのメリットが考えられるため、この方式とする。

【0181】各トランザクションは開始時に割り振られたログバッファにログを作成し、トランザクション終了時にまとめてログライトバッファへコピーする。ログライトバッファの内容を実際にI/O出力するのはログライトデーモン105と呼ぶデーモンの働きによる。

【0182】巨大トランザクションには、一般トランザクションが用いるログバッファよりも大きいログバッファ（以降、それぞれを一般ログバッファ、巨大ログバッファと呼ぶ）を1つ用意する。巨大トランザクションも当初は一般ログバッファの1つを用いる。巨大トランザクションとログバッファの関係には次の二段階がある。

1) あまり多くのメタデータを更新せず、一般ログバッファで十分足りると判断できた時点で、次の巨大トランザクションのBEGINを受け付ける。

2) 更新するメタデータ数がある程度多く、一般ログバッファでは足りないとは判断できた時点で、巨大ログバッファにこれまでの内容をコピー、そこで処理を継続する。

3) 更新するメタデータ数が大変多く、巨大ログバッファでは足りない場合には、サブトランザクションとして分割する。

【0183】2)におけるバッファ間のコピーが負担となりそうであるが、巨大トランザクションがそれほど多くないと考えると、あまり頻繁に発生するものではないので、この方法とする。

【0184】一般ログバッファは状態（利用中・未利用）について、ビットマップで管理される。各ログバッファを管理する構造をログバッファ数の配列で確保し、それぞれが

- ・ログバッファのアドレス
- ・これまでに採取したログサイズを持っている。

【0185】また、ログバッファの内容は専用のログライトデーモン105によってログボリューム120へ反映するが、そのログライトデーモン105が複数のログバッファの内容を一括してI/O発行するために、トランザクション終了時にそれぞれのログバッファの内容をログライトバッファ150と呼ぶ専用の領域にコピーする。

50 【0186】ログライトバッファを管理する構造とし

て、

- ・追加モードにあるログライトバッファはどちらか
 - ・それぞれのログライトバッファに溜められたログサイズ
- がある。

【0187】ログライトバッファの内容をログボリューム120へ書き出している間は、その後のトランザクションが終了するためにログライトバッファにログバッファをコピーしようとしても、I/O中であるためにガードしなければならない。これは性能劣化に繋がるため避けなければならない。そのため、ログライトバッファは2本用意する。

【0188】一般トランザクションはトランザクション開始時に、空のログバッファをビットマップより見つけ（未利用ログバッファを0で示し、ここで1とする）、そのビット番号がトランザクション番号となる。

【0189】巨大トランザクションが一般ログバッファにて処理進行中、一般ログバッファでは入りきらないと判断されるとこれまでの内容をこの巨大ログバッファへコピーし、そこで処理を継続する。この時、これまで用いていた一般ログバッファは空きバッファリストへ繋ぎ直される。

(2.2) 巨大トランザクションの管理

空き領域管理ツリーや獲得済領域管理ツリーを操作するトランザクションを巨大トランザクションと定義する。巨大トランザクションは場合によっては木構造を大きく変更しなければならないかもしれない。この時、サブトランザクション化する必要が生ずる。

【0190】サブトランザクション化した巨大トランザクションが並行動作した場合、どれでも多くのメタデータを更新すると、メタキャッシュ130がいずれ全てダーティとなり、新たにメタデータを読み込んで更新しようにも追い出すこともできずにデッドロックに陥ることが考えられる。

【0191】そのため、巨大トランザクションについては並行動作ができないよう、ここではシリアルライズした。しかし、上記の通り、多くのメタデータを更新する可能性があると考えられるトランザクションは多い。これらを全てトランザクション終了までシリアルライズすると、その性能インパクトは大きい。

【0192】巨大トランザクションも、場合によってはそれほど多くのメタデータを更新しないこともありえる。一般トランザクションと同等の数しかメタデータを更新しないのであれば、扱いを一般トランザクションと同様にすることによって、その性能インパクトを小さくすることが可能である。

【0193】そこで、巨大トランザクションが一般トランザクションと変わらない程度しかメタデータを更新しないと分かった時点で、この巨大トランザクションの扱いは一般トランザクションと同じにする（デグレー

ド）。

【0194】巨大トランザクションはその開始時には一般ログバッファの1つを一般トランザクション開始時と同様に与えられる。しかし、1つの巨大トランザクションが動作中は次の巨大トランザクションにログバッファを与えない点が一般トランザクションの場合と異なる。

【0195】巨大トランザクションがある程度以上のメタデータを更新することが分かった時点で、これまでの一般ログバッファの内容を巨大ログバッファへコピーし、そこで処理を継続する。

【0196】しかし、それほど多くのメタデータを更新しないと判断されれば、そのまま一般ログバッファで処理が継続され、また、次の巨大トランザクションに対し、処理の開始（ログバッファに使用）を許可する。

【0197】こうして、巨大トランザクションを途中から一般トランザクションとして扱うこと（デグレード）を実現する。

(2.2.1) デグレード契機・サブトランザクション化契機

巨大トランザクションは複数のサブトランザクションに分割される場合もあれば、一般トランザクションへデグレードする場合もある。それぞれの判定基準は以下の通りである。

◎デグレード判定基準

・一般ログバッファの残りサイズで、将来全てのメタデータ更新が完了できると判断できた場合。

◎巨大トランザクション用ログバッファへの移行契機

・一般ログバッファの残りサイズが次のステップの最大変更量よりも少ない場合。

◎サブトランザクション判定基準

・巨大ログバッファの残りサイズが次ステップの最大変更量よりも少ない場合。

(2.3) 一般トランザクションのログ採取

一般トランザクションは、以下の手順に従いログを採取する。

1) LOG_BEGIN

a) 利用するログバッファを確定する。

【0198】b) ログバッファの先頭にBEGINマークを作成する。

2) 各メタデータの更新

c) 更新されたメタデータがリリースされる（ロックが解放される）直前に、更新されたメタデータについて、ヘッダ（更新情報）をログバッファに作成し、更新内容をログバッファにコピーする。

【0199】d) 更新されたメタデータをリストに繋ぐ。

3) LOG_END

e) ENDマークをログバッファに作成する。

【0200】f) ログバッファをログライトバッファへコピーする。

g) ログライトデーモン 105 がログボリューム 120 へ反映する。

h) このトランザクションで立てられた追い出し不可フラグを落とす。

(2. 3. 1) LOG_BEGIN

トランザクション開始を意味する。同一関数内に LOG_END の宣言が必要。並行動作可能数だけ既にトランザクションが動作していたら、それらのどれか 1 つが終了するまで寝て待つ。

【0201】 a) 現在実行中のトランザクション数がログバッファの数と等しい場合、それは既に許された並行動作数だけトランザクションが実行中であることを意味する。そのため、他トランザクションが終了するまで寝る。そのような状態でなければ、ログバッファの利用状況を管理するビットマップより未利用状態のログバッファを 1 つ選び出し、そのビットを立てる。さらに、並行動作数カウンタをインクリメントする。

【0202】 b) 前述の BEGIN マークをログバッファの先頭に作成する。

(2. 3. 2) 各メタデータの更新

メタデータを参照した場合にはログ採取の必要はない。更新した場合のみログを採らなければならない。

【0203】 c) メタキャッシュ 130 に読み込んだメタデータについて処理が終了し、メタキャッシュ 130 より解放するための関数（リリース関数）を呼び出す際、「更新した」ことを明示された場合、ログを採取する。

【0204】 i ノードについてはログ対象トランザクション内でのロック解放時がログ採取契機である。具体的には、上記で使用权を得たログバッファにヘッダを作成し、メタデータの更新内容をコピーする。このとき、メタデータの種類によってその手法が若干異なる。

【0205】 c-1) ビットマップの場合：ヘッダには獲得（解放）したメタデータ番号を入れ、更新内容は「0→1」「1→0」とビット操作だけとする。

c-2) メタデータ本体の場合：更新内容にはメタデータをそのままコピーする。

【0206】 c-3) スーパーブロックの場合：巨大トランザクションがデグレードした場合のみ、一般ログ内でのスーパーブロックのログ採取がありうる。その時刻（シーケンシャル番号）とデータ空きサイズを記録する。

【0207】 コピーするとともに、ログバッファのターミナルで管理されるログサイズに、ここで作成したログのサイズを加える。

d) メタキャッシュ 130 の大きさは有限であることから、トランザクションが進行するためには、必要のないメタデータをメタキャッシュ 130 から追い出し、その場所に必要なメタデータを読み込むという処理を行う追い出し機構がある。更新されたメタデータはログを書き出すまでは追い出されてはならない。そのために、この

メタデータをメタライトリストと呼ぶ、メタボリュームへの反映を司るメタライトデーモン 104 が参照するリストへ繋ぐ。

(2. 3. 3) LOG_END

トランザクションの終了を示す。ここで、END マークの作成を行う。ログボリューム 120 反映はログライトデーモン 105 による作業である。

【0208】 e) LOG_BEGIN にて作成した BEGIN マークに対応する END マークをログバッファの末尾に作成する。

f) 利用したログバッファの内容を追加モードにあるログライトバッファにコピーする。この時にログ番号を決定し、BEGIN マーク、END マークに埋め込むログバッファのターミナルに記録していたログのサイズをログライトバッファのサイズに加える。同期書き込み要求の場合は、ログライトデーモン 105（詳細後述）が正常にログボリューム 120 に反映したことを待つ必要があるが、同期書き込み要求でない場合には、ログバッファを解放（ビットを落とす）して終了する。

【0209】 g) 後述するログライトデーモン 105 が、ログライトバッファの内容をログボリュームへ反映する。

h) リリース時に立てた追い出し不可フラグを全て落とす。ただし、フラグを落とすのはログライトデーモンによってなされるため、このトランザクションはそのまま終了する。

(2. 4) 巨大トランザクションのログ採取

巨大トランザクションも、一般トランザクションとほぼ同様の処理だが、最大の違いは、トランザクションの状況によってログバッファをサイズの大きい巨大ログバッファへ移行したり、サブトランザクションとして分割したログ採取を行わなければならない場合があることである。

1) LOG_BEGIN

a) 利用するログバッファを確定する。

【0210】 b) ログバッファの先頭に BEGIN マークを作成する。

c) BEGIN マーク作成と同時に、オペレーションログをログバッファに作成する。

2) 各メタデータの更新

メタデータがリリースされた時に「更新したこと」が明示されたら、

d) 更新メタデータが空き管理情報の場合、更新メタデータがリリースされる度に BTF リストあるいは BTA リストとよぶ空き管理メタデータのために用意するリストに繋ぐ。この時、既にリストに繋がれていればリスト構造には手を加えない。

【0211】 e) 更新メタデータが空き管理情報以外の場合、更新されたメタデータがリリースされる度にヘッダをログバッファに作成し、更新内容をログバッファ域

にコピーする。

【0212】f) 更新メタデータをリストに繋ぐ。まだ一般ログバッファで処理している場合、

g) 一般ログバッファで足りるか判定する

g-a) 足りないなら巨大ログバッファへこれまでの内容をコピー。

【0213】g-b) 足りると確定でき、かつ、デグレード条件を満たせば、次の巨大トランザクションを受け付ける。

g-c) まだ判断ができなければそのまま継続する。既に巨大ログバッファで処理している場合、

h) サブトランザクション化するか判定する

h-a) するなら、BTFリスト及びBTAリストをたどりながらそれらをログバッファにコピー、ENDマークを作成し、ログライトバッファへコピーする。ログライトデーモン105を起動し、ログを出力する。

【0214】h-b) まだしないなら、そのまま継続する。

3) LOG_END

i) 最終ENDマークをログバッファに作成する。

【0215】j) ログバッファをログライトバッファへコピーする。

k) ログライトデーモン105がログボリューム120へ反映する。

l) このトランザクションで立てられた追い出し不可フラグを落とす。

(2.4.1) LOG_BEGIN

巨大トランザクションも最初は一般ログバッファを用いる。

【0216】a) 一般トランザクションの場合は、現在の並行動作トランザクション数とログバッファの数との関係によってログバッファを獲得できるかどうか定まった。巨大トランザクションの場合は、現在他の巨大トランザクションが動作中か否かが問題であり、一般トランザクションの並行動作数とは関係しない。巨大トランザクションが動作中か否かのフラグを調査し、非動作中であれば、そのフラグを立て、ログバッファの利用状況を管理するビットマップより未利用状態のログバッファを1つ選び出し、そのビットを立てる。巨大トランザクションが現在動作中であれば、終了まで寝て待つ。

【0217】b) BEGINマークを作成する。ここで、トランザクションタイプに巨大トランザクションとなりうるトランザクションが指定されていた場合には、必ず後ろにはオペレーションログが付随する。対応するENDマークがないログはリプレイしてはならない。また、サブトランザクション化しているのに、最終ENDマークがないなら、オペレーションログのリプレイが必要である。

【0218】c) 巨大トランザクションをサブトランザクションログとして分割する場合にはオペレーションロ

グを採取する必要がある。ここで、オペレーションログとは、各ログ採取対象関数に与えられたパラメータが、将来リプレイすることが可能な形に変更されたものである。具体的には、メモリアドレスで与えられるパラメータは、メタボリューム内のオフセットに変更して記録する。

(2.4.2) 各メタデータの更新

d) 更新メタデータが空き管理情報である場合、一回のトランザクションで同一の領域が何度も変更される場合が考えられる。その都度、ログを採取しているとインコアログやログボリュームがフルになる可能性が高まる。これを回避するために、更新情報をリストによって管理し、複数回更新された空き管理メタデータを1つにまとめてログに記録する。具体的には、メタデータの状態毎にリスト管理する専用のBTFリスト及びBTAリストに繋がれる。このリストに繋がれていれば、そのデータはダーティであることを意味する。初めて更新するデータであれば、ターミナルから繋がるリストに追加する。既にリストに繋がれているのであれば、2回目以降の更新であることを意味し、ポイントについてはそのまま良い。リストにメタデータを追加する場合には、ログサイズを更新する。

【0219】e) 更新メタデータが空き管理情報以外であれば、同一メタデータに対して何度もホールド/リリースが発行されるとは考えられないので、リリースの度(ロック解放の度)にログバッファへコピーする(一般トランザクションの場合と同じ)。ログサイズを更新する。

【0220】f) 既に何度か述べたが、更新されたメタデータがログよりも先にメタボリューム111~113へ反映されてはならない。この追い出し不可状態もリスト構造によって管理される。

【0221】g) この巨大トランザクションが更新するメタデータが、以降、どれだけの数のメタデータを更新するかを算出することによって、処理が分かれる。

g-a) 現在利用している一般ログバッファの残りでは次ステップの実行によるメタデータの更新の全てをまかないきれない場合があると判断した場合には、巨大ログバッファへの移管を行う。

【0222】g-a-1) これまでに溜まったログバッファの内容を巨大ログバッファへコピーする。この時、BTFリストはそのまま繋いだままとし、BTAリストは巨大トランザクション用のターミナルへ移動する。

【0223】g-a-2) 巨大ログバッファを利用して、その巨大トランザクションは処理を継続する。

g-b) 現在利用している一般ログバッファの残りだけで、以降のメタデータ更新を全て記録できると判断できるならば、この巨大トランザクションは一般トランザクションにデグレードする。BTFリストにメタデータが繋がれている場合はデグレード条件を満たさないため、

ここでの処理はありえない。BTFリストはそのまま利用を続ける。

【0224】gb-1) 巨大トランザクションが動作中か否かを示すフラグを落とす。

gb-2) 一般トランザクションの並行動作数カウンタをインクリメント。

gb-3) 次の巨大トランザクションが寝ているならば起こして、実行を受け付ける。

【0225】gb-4) このまま一般ログバッファを利用して処理を継続する。

g-c) まだ判断できないのであれば、このまま一般ログバッファを利用して処理を継続する。

【0226】h) サブトランザクション判定基準(前述)に基づき、このトランザクションをサブトランザクション化するかどうか判定する。

h-a) サブトランザクションに分割する場合は、

h-a-1) 更新された空き管理情報を、リストを辿りながらログバッファへコピーする。

【0227】h-a-2) ENDマークを作成する。

h-a-3) ログライトバッファへコピーする。

h-a-4) ログライトサイズを更新する。

【0228】h-a-5) ログライトデーモン105を強制起動する。

h-a-6) ここで変更した全てのメタデータの追い出し不可フラグを落としてメタライトリストに繋ぐ。これにより、メタライトデーモン104は任意の時にこれらのメタデータを書き出すことが可能となる。

【0229】h-a-7) 巨大ログバッファの先頭にBEGINマークを作成する。

h-a-8) オペレーションログを作成する。

h-b) サブトランザクションに分割する必要がない間は、巨大ログバッファ内でそのまま継続する。

(2.4.3) LOG_END

トランザクションの終了を示す。ここで、ENDマークの作成を行う。実際のログボリューム120反映はログライトデーモン105の仕事である。

【0230】i) 最終ENDマークを作成する。最終ENDマークは通常のENDマークとマジックワードが異なるだけである。

j) 利用したログバッファの内容を追加モードにあるログライトバッファへコピーする。この時にログ番号を決定し、ログサイズをログライトサイズに加える。同期書き込み要求の場合は、ログライトデーモン105が正常にログボリューム120反映したことを待つ必要があるが、その他の場合には待たないで次の処理へ進む(すなわち非同期書き出しである)。巨大トランザクション動作中フラグを落とし、寝て待つ次の巨大トランザクションを起こす。

【0231】k) ログライトデーモン105が、ログライトバッファの内容をログボリュームへ反映する。

1) 更新したメタデータをログバッファへコピーした時に立てた追い出し不可フラグを全て落とす。ただし、フラグを落とすのはログライトデーモン105によってなされるため、このトランザクションはそのまま終了する。

(2.5) ログライトデーモン

並行して動作し、順次終了するトランザクションによって更新されたメタデータのログは専用の書き出しデーモン(ログライトデーモン105)によってログボリュームへ反映する。このデーモンはマウント時に起動され、アンマウント時に停止する。すなわち、ファイルシステム毎にスレッドを生成する。

【0232】各トランザクションが終了すると、ログバッファ内にENDマークが作成され、このログバッファの内容はログライトバッファ150にコピーされる。このログライトバッファ150は複数のログバッファの内容を一括してログボリューム120に反映するためのものであり、そこにはメモリコピーの負担があるが、何度もI/Oを発行するよりは良いと考える。

【0233】ログライトバッファ150へコピーされると、そのログバッファは利用中バッファリストから空きバッファリストへと繋ぎ直され、かつ、トランザクションが同期書き込み要求でなければ、動作中トランザクション数をデクリメントする。これにより、次トランザクションの実行が進むようになる。

【0234】ログライトデーモン105はログライトバッファの内容を、定期的、あるいは同期書き込み指定のトランザクションがある場合などにログボリュームに反映する。反映している間(I/O中)は2本あるログライトバッファのうち、もう片方のログライトバッファに内容をコピーしてそのトランザクションは処理を終了する。

(2.5.1) 処理手順

ログライトデーモン105単体の処理手順はそれほど複雑ではない。

【0235】a) 現在のログライトバッファをライトモードにし、もう片方を追加モードにする。

b) ログライトバッファの内容をログボリューム120へ同期書き出しする。

【0236】o) 場合に応じて有効範囲情報を書き出す。

d) ログ待ちリストに繋がるメタデータを、メタライトリストへ繋ぎかえる。

e) 規定時間の間、スリープする。

【0237】f) 規定時間が経過、または他の要因で起こされたらa)へ。

以下、詳細説明である。

a) I/O中はその対象域に対して追加更新は許されない。そのため、現在のログライトバッファが書き出しが終わり、次の書き出しが始まるまでは、もう片方のログ

ライトバッファへ終了したログバッファをコピーするよう設定する。I/O中のログライトバッファをライトモード、ログバッファの内容をコピーする方を追加モードと呼ぶ。切り替えはこのデーモンによって行われ、各トランザクションは常に追加モードにあるログライトバッファに自ログバッファの内容をコピーする。

【0238】b) ログライトバッファに新しい内容が含まれていないのであれば、I/Oを発行する必要はない。そこで、まずログライトサイズを調べ、ゼロであればI/Oを発行せず、タイマを指定して再び寝る。書き出すべきログが存在するのであれば、以下の手順に従う。

【0239】b-a) ログボリュームの空きサイズ判定。

ログボリュームの先頭オフセット (A)、メタライトリスト先頭のメタデータを管理する構造の上書き可能オフセット (B) を調べる。それとログライトオフセット (C)、及びログボリュームの最終オフセット (D) から以下の手順となる。

・ $A < B \leq C < D$

①ログライトサイズが $(D - C)$ 以下ならば、Cから書き出す。

②ログライトサイズが $(D - C)$ より大きく、かつ、 $(B - A)$ 以下ならば、Aから書き出す。

・ $A \leq C < B \leq D$

③ログライトサイズが $(B - C)$ より小さければ、Cから書き出す。

【0240】これらの条件に合致せず、ログの出力ができない場合には、メタライトデーモン104を起動して、自分はスリープする。メタライトデーモン104の動作により起動されたら、再度、空きサイズ判定から行う。

【0241】b-b) ログボリューム120へ同期書き出しする。

b-c) エラー判定。同期書き出し後、エラー判定を行う。エラーが発見された場合の処置については後述(2.5.3)する。

【0242】b-d) 領域判定。

書き出しが終わった後、再び、b-a)と同様な空き領域の判定を行う。ここで、残りが少ないと判定された時、その残りの大きさに応じてメタライトデーモン104を「平常起動」「緊急起動」する。

【0243】c) ファイルシステム復元に必要なログは、メタライトリスト先頭の上書き可能オフセットからたった今書き出した位置までである。そこで、それを有効範囲情報としてディスクへ書き出す。ここで、書き出しはログのシーケンシャル性を考慮し、ある程度の間隔をおいて行う。ここでは、回数に応じて、数回に一回、書き出すこととした。

【0244】d) b)にて書き出したログに含まれるメ

タデータは全て「ログ未反映状態」、すなわち追い出し不可状態としてリスト(ログ未反映リスト)に繋がれているはずである。そこで、ログ未反映リストを辿りながら、そこに繋がるメタデータをメタライトリストに追加することにより、メタライトデーモン104が任意の時にこれらのメタデータをメタボリュームに反映できるようになる。

【0245】e) ログもシステム全体から見れば非同期書き出しとし、トランザクションが同期書き込み要求でなければ、ログを書き出す前にそのトランザクションは終了することができる。さらにI/O発行回数を削減するために、複数のトランザクションを一括して書き出すために、しばらくの間スリープし、その間にログライトバッファへ複数トランザクションのログを溜める。

【0246】f) 規定時間後には自分で起きて、処理を繰り返す。しかし、他要因によって突然起こされて処理を始める場合もある。「その他の要因については後述(2.5.2)する。

(2.5.2) 動作契機

20 ログライトデーモン105は以下の契機でログライトバッファの内容をログボリュームへ書き出す。

◎定周期

ログライトデーモン105は一定周期毎にスリープ状態から自発的に目覚め、ログライトバッファ150の内容をログボリューム120へ反映する。

◎同期書き込み要求のトランザクション

トランザクションによっては同期書き込み要求で呼ばれる場合がある。このトランザクションについては、ログの書き出しを待たなければ終了してはならない。そのため、LOG_END呼び出し後に、明示的にLOG_SYNCを行う。LOG_SYNCはログライトデーモン105が寝ている場合には起こし、ログライトバッファの内容をその場で書き出させる。

【0247】現在ログライトデーモン105がI/O中であったら、ログライトデーモン105の処理が終わるのを寝て待つ。

◎ログライトバッファ不足定

周期毎にログライトバッファをクリアしても、大きなログバッファが連続して終了すると、その前にログライトバッファがフルに近い状態になることがありえる。この時にはログライトデーモン105が起こされ、ログライトバッファ150の内容をログボリューム120に書き出してクリアする。

【0248】具体的には、あるトランザクションが自ログバッファの内容をLOG_ENDによってコピーしようとした時、ログライトバッファ150の残りサイズが自ログよりも小さいと判断された時、ログライトデーモン105を起こし、追加モードのログライトバッファ150を切り替えてもらい、その間は寝て待つ。

【0249】現在I/O中であり、かつ追加モードのロ

グライトバッファ 150 が足りなくなった時には、その
トランザクションは待ち合わせざるを得ない。

◎メタキャッシュ不足

トランザクション実行中に、メタキャッシュ 130 域の
いずれかのメタデータを追い出さなければ必要なメタデ
ータをメタボリューム 111~113 から読み込むこと
ができずに処理が進まなくなる場合が考えられる。

【0250】そのため、トランザクションが現在キャッ
シュ上のメタデータを追い出そうとする時、メタキャッ
シュ 130 上のメタデータが全てダーティであり、かつ、
ログ待ちリストに繋がれたメタデータが存在する場合に
は、ログライトデーモン 105 を起動する。

◎アンマウント時

アンマウント時には必ず書き出さなければならない。そ
のため、アンマウント処理の延長でログライトデーモン
105 を起こし、書き出しを行う。この時、アンマウン
トを実行するため、該当ファイルシステムにメタデータ
を更新するようなトランザクションは動作していないこ
とが保証される。従って、現在のログライトバッファの
内容を書き出せば、それで全てである。

【0251】ログリアとバッファの内容を書き出した
後、ログライトデーモン 105 は終了する。

(3) メタデータ管理

次に、メタデータの管理方式について説明する。メタキ
ャッシュ 130 内のメタデータは `metalist` 構造
体によって管理される。`metalist` 構造体には以
下のメンバがある。

- ・メタデータへのポインタ
- ・TRANS (トランザクション) リストポインタ
- ・メタライトリスト `prev` ポインタ
- ・メタライトリスト `next` ポインタ
- ・ログ待ちリスト `prev` ポインタ
- ・ログ待ちリスト `next` ポインタ
- ・ログシーケンス番号
- ・ログボリューム内オフセット
- ・トランザクション番号
- ・状態フラグ
- ・メタデータタイプ
- ・`buf` 構造体実体

(3.1) メタデータの状態遷移

`metalist` 構造体には次の 6 種類の状態があり、
4 種類のリストに繋がれる。それは `metalist` 構
造体のフラグによって示され、それぞれターミナルを別
とするリストに繋がれることによって実現する。全ての
`metalist` 構造体はいずれかのリストに繋がっ
ており、例外はない。

A) 空き管理構造更新状態

データ空き領域を管理するメタデータがトランザクシ
ョンによって更新されると、リストに繋ぎ、将来まとめて
ログバッファにコピーすることは既に述べた。トランザ

クション終了時に本リストに繋がれたメタデータが直接
ログボリュームへ反映される。リストは並行動作数に応
じて必要である。このリストを BTF リストと呼ぶ。

【0252】この状態にあるメタデータはまだログバッ
ファへはコピーされておらず、同一トランザクションに
よって再度更新される場合がある。既にリストに繋がれ
ているメタデータであった場合は、リスト状態は変更し
ない。当然、メタボリュームに反映してはならない。

【0253】BTF リストは単方向環状リストであり、
トランザクション途中状態と同じメンバを用いて繋がれ
ている。

B) 利用域管理構造更新状態

実施例ではファイルシステムをエクステント管理してい
る。i ノードから導かれる、そのファイルが利用してい
るエクステントを示す、間接エクステントブロックにつ
いても、トランザクション内で 1 つの間接エクステント
ブロックが何度も更新される場合が考えられる。そこ
で、空き管理構造の場合と同様に、トランザクション中
は独自のリストに繋ぎ、トランザクション終了時にまと
めてログバッファへコピーする。リストは並行動作数に
応じて必要である。このリストを BTA リストと呼ぶ。

【0254】この状態にある間接エクステントブロック
はまだログバッファへはコピーされておらず、同一トラ
ンザクションによって再度更新される場合がある。既に
リストに繋がれている間接エクステントブロックであっ
た場合は、リスト状態は変更しない。当然、メタボリュ
ームに反映してはならない。

【0255】BTA リストは単方向環状リストであり、
トランザクション途中状態と同じメンバを用いて繋がれ
ている。

C) トランザクション途中状態

トランザクション途中を示す。この状態のとき、該当す
るメタデータを更新したトランザクションは、ある 1 つ
のログバッファを専有しており、既にそこに更新後状態
がコピーされている。リストは並行動作数に応じて必要
である。このリストを TRANS リストと呼ぶ。

【0256】しかし、まだトランザクションが終了して
いないことからログがログボリューム 120 へ反映され
ておらず、本メタデータもメタボリュームへの反映はで
きない。この状態にあるメタデータはファイルのロック
によって保護されているため、他トランザクションから
参照・更新されることはない。

【0257】TRANS リストのターミナルは配列にな
っており、その要素番号はトランザクションが用いてい
るログバッファの番号 (トランザクション番号) に対応
する。

【0258】メタライトリストやログ待ちリストに繋が
れているメタデータが繋がる場合がある。TRANS リ
ストは単方向環状リストであり、BTF リストが用いる
メンバを併用する。

【0259】D) ログ待ち状態

トランザクションは既に終了しているが、ログバッファの内容はログライトデーモン105によってログボリューム120へ反映されるため、この時点ではまだログボリューム120に反映されていない状態である。(デーモンは同期ライトを行うが、トランザクションの視点から見れば、ログ反映が非同期に行われているように見える。)

トランザクションが終了する際に、C)の状態にあるTRANSリスト(巨大トランザクションの場合はBTFLリストも)をログ待ちリストに繋ぎかえる。このリストはトランザクションが終了する毎に後方へ伸びるリストであり、ログライトバッファと同数の2本存在する。

【0260】この状態にあるメタデータは既にトランザクションが終了しており、ログはログライトバッファにコピーされている。トランザクションが終了していることから、他トランザクションによって参照・更新される可能性がある。この時(他トランザクションによって状態が変化した時)、リスト位置は変更せず、状態フラグだけを変更する。

【0261】メタライトリストに繋がれたメタデータが、他トランザクションによって再度更新されたために、トランザクション途中状態になり、そのトランザクションが終了したことによって、ログ待ち状態になった場合には、このメタデータはメタライトリストにも繋がれている。

【0262】ログライトデーモン105によって、ログの反映が進むと、それに応じたメタデータをメタライトリストへ追加していく。この時、既にメタライトリストに繋がれているメタデータは、そのままの位置でいなければならない。

E) 書き出し可能状態

これは、ログバッファのログボリュームへ反映が終了し、自由にメタデータをメタボリュームへ反映することができるようになった状態である。この状態にあるメタデータは他トランザクションによって参照・更新される可能性がある。この時、リスト位置は変更せず、状態フラグだけを変更する。

【0263】繋がるリストはメタライトリストであり、1本だけ存在する。メタライトデーモン104はこのリストを参照してメタボリュームへの反映を行う。

F) I/O中状態

この状態にあるメタデータは現在非同期ライトによってメタボリュームに対してI/Oを投げているが、まだ完了していない。I/O途中であるため、他トランザクションは操作することができない(参照することはできる)。メタライトリストにそのまま繋がれている。

(3-2) ログボリュームの上書き判定情報

ログボリュームはサイズが有限であるため、サイクリックに利用し、過去のログを上書きしてシステムは動作す

る。ここで、過去のログを上書きするためには、そのトランザクションが更新したメタデータが全てメタボリュームに反映されていることが必要である。上書き可能領域を管理するために、以下の構造を設け、メタライトデーモン104が変更、ログライトデーモン105が参照する。

◎ログシーケンス番号

各トランザクションが終了し、ログバッファの内容をログライトバッファへコピーする毎に与えられる、シーケンス番号である。ログボリューム内のBEGINマーク、ENDマークに含まれる番号と同一である。既にライトリストに繋がるメタデータが再度更新される場合にも、この番号は変更されない。

◎ログボリューム内オフセット

ログボリューム内にはトランザクション毎のログが連続して並んでおり、上書きするためには、それぞれのトランザクションが更新した全てのメタデータがメタボリュームへ反映されている必要がある。

【0264】同一のログ番号を持つmetalist構造体は、ここにはそのログの先頭オフセットを挿入する。LOG__ENDによってログバッファの内容がログライトバッファにコピーされる際、そのアドレスとログボリュームの書き出しオフセットから導かれる。トランザクション毎のログボリューム内オフセットがTRANSリストを辿りながらそれぞれのmetalist構造体に設定する。

【0265】ログライトデーモン105は、ログを書き出す際にメタライトリストの先頭に繋がるmetalist構造体を参照し、現在のポインタにログライトバッファに入っているログサイズを足した位置が、このオフセット値を超えないことを確認した後で書き出しを行う。

(3.3) メタボリュームへの反映

トランザクションによって更新されたメタデータは、トランザクションが終了しログがログボリューム120に反映されるまでは書き出されてはならない。そのために、メタキャッシュ130上の各メタデータはそれぞれの状態に応じてリストに繋がれ、唯一メタボリュームへの反映が許可されているリストはメタライトリストである。

【0266】時間のかかるI/O要求をトランザクション内で行うことを避けるために、メタデータのメタボリュームへの反映はトランザクションとは関係ないところで動作するデーモンに委ねる。このデーモンはメタライトリストだけを意識し、metalist構造体内の情報から状態に応じてI/O発行するかどうかを決定し、関連付けられたbuf構造体を用いてメタデータをメタボリュームへ反映する。

【0267】デーモンはメタライトリストに繋がれたメタデータを書き出した後、そのメタデータをリストから

外し、cleanであると設定する。

(3.4) メタライトデーモン

メタライトデーモン104は、マウント時に起動され、アンマウント時に停止する。すなわち、ファイルシステム毎にスレッドを起動する。

【0268】 ログ書き出しの終わったトランザクションが更新したメタデータは、それを管理するmetalist構造体が全てメタライトリストに繋がれている。メタライトデーモン104はメタライトリストに繋がるメタデータを、ログボリュームの残りが少なくなったと判断された場合などに非同期でメタボリュームに反映する。反映している間は該当するメタデータは更新することができず、I/O終了を待ち合わせることになる。

【0269】 メタライトデーモン104はメタライトリストを意識し、繋がるメタデータをメタボリュームへ反映する。これにより、ログボリュームの上書き可能領域が拡大する。

【0270】 しかし、メタライトリストに繋がれているメタデータの中にも、再度トランザクションによって更新が進み、メタボリュームへの反映が禁じられているものが存在する場合がある。この時、他のメタデータを書き出すとしても、そのメタデータについては非同期ライトの発行を待ち合わせなければならない。このようなメタデータが存在すると、以降のメタデータを全て書き出せたとしても、上書き可能領域を拡大することができない。

【0271】 メタライトデーモン104は他者から起動されて処理を開始する場合と、自発的に起動する場合がある。以下に処理手順を述べるが、システムの状態（ログボリュームの空きやメタキャッシュ130の空きなど）に応じて動作が若干異なる。また、ここで述べる処理手順にはデーモン内だけではなく、ドライバから呼ばれる関数での処理も含まれている。

(3.4.1) 自発起動処理

メタライトデーモン104はタイマによって自発的に起動して、それまでにメタライトリストに繋がれた書き出し可能なメタデータをメタボリュームに反映する。この時、メタライトリストの全てを書き出す訳ではなく、ある一定の数だけ非同期書き出しを行う。普段から少しずつメタデータの書き出しを行っておくことによって、資源不足による起動を避け、I/O負荷分散を図る目的がある。

【0272】 a) 自発起動時には、メタライトリストに繋がるメタデータ数を調査する。

b) メタライトリストの先頭から、メタデータの状態を調べる。

c) メタデータが書き出し可能状態であれば、その状態をI/O負荷態に変更し、非同期ライトを発行する。

【0273】 d) b__iodoneの関数がI/Oの成功を確認する。

e) b__iodoneの関数がメタライトリストから外す。

f) 規定数だけ繰り返す

g) スリープする。

【0274】 以下、詳細説明である。

a) 自発起動間隔として定義する間隔毎にメタライトリストに繋がるメタデータ数を確認する。具体的には、メタライトリストのターミナルに含まれる、リンクされたメタデータ数を調べる。このとき、それほど多くのメタデータが繋がれていない場合には書き出しは行わない。

【0275】 b) メタライトリストには基本的にはログの書き出しが終わった、書き出し可能状態のメタデータが繋がれている。しかし、トランザクションが終了してメタライトリストに繋がれた後で他トランザクションによって更新されると、そのメタデータだけは書き出し不可の状態に戻ってしまう。そのため、メタライトデーモン104はメタライトリストに繋がるメタデータの状態を確認しながら書き出し処理を行わなければならない。

【0276】 c) 現在ポイントするメタデータが書き出し可能状態であれば、その状態をI/O状態に書き換え、metalist構造体に含まれるbuf構造体を用いてI/Oを発行する。I/O中状態のメタデータは他トランザクションから更新されることはない。メタデータのディスクライトは非同期ライトで行い、デーモンはI/Oの結果を待たずに次の処理へ進む。

【0277】 メタデータが書き出し不可状態であるなら、そのメタデータは諦め、リストの次に繋がるメタデータに進み、状態を調べる。

d) 非同期ライトによって発行されたI/Oの結果を判定するのは、buf構造体のメンバb__iodoneに組み込まれた関数である。この関数にはbuf構造体が引数に与えられ、それを元にエラー判定処理を呼び出す。ここでエラーを検出した場合には、異常系処理へ進む（後述）。成功していた場合には、該当メタデータのclean数をインクリメントする。

【0278】 e) b__iodoneに組み込まれた関数はメタライトリストの管理も行う。引数に渡されたbuf構造体の上を見るとそこはmetalist構造体になっており、そのフラグからI/O中フラグを落とし、メタライトリストから外す。また、メタデータをダーティ状態からclean状態へ変更する。これは、メタキャッシュ130で管理されるメタデータである場合には、それぞれの管理構造体で管理されており、この管理構造体はmetalist構造体からポイントされる。

【0279】 全ての処理が終了したら、このmetalist構造体をリリースする。

f) メタライトリストに繋がる書き出し可能メタデータを、次々とリストを辿りながら定義した数だけ処理を繰り返す。ここで、書き出しが不可とされているメタデータについては、この数には含めない。また、メタライト

リストが定義数よりも少ない場合には、当然そこで終了となる。

g) スリープする。再度、自発的に起動するか、あるいは資源不足によって他から起動されるまで、スリープは継続する。

(3.4.2) 平常処理

システム状態が以下の場合には、ここで述べる処理手順に従いメタライトデーモン 104 は動作する。

◎ログボリュームの残りが少ない。

【0280】 ログライトデーモン 105 が判断する。ログライトバッファを書き出す際に、上書き可能域の大きさを計算し、この閾値を下回った時にメタライトデーモン 104 を起動する。

◎メタキャッシュの残りが少ない。

【0281】 更新リリースがあった時、各メタデータの clean 数がデクリメントされるが、その数がこの閾値を下回った時にメタライトデーモン 104 を起動する。

a) メタライトリストの先頭から、メタデータの状態を調べる。

【0282】 b) メタデータが書き出し可能状態であれば、その状態を I/O 中状態に変更し、非同期ライトを発行する。

c) b__i o d o n e の関数が I/O の成功を確認する。

【0283】 d) b__i o d o n e の関数がメタライトリストから外す。

e) メタライトリストの最後まで繰り返す。

f) スリープする。

【0284】 以下、詳細説明である。

a) ~ d) 自発的に起動した場合と同じである。

e) メタライトリストを辿りながら、繋がる全ての書き出し可能メタデータについて処理を繰り返す。平常処理時には、書き出し不可のメタデータについては飛ばして処理を行う。そのため、ログボリューム不足時には、メタライトリストの先頭、すなわち、ログボリュームの上書き可能位置を制限しているメタデータが書き出せなければ資源不足は解消しないが、平常処理であるため、ここでは特別な対処を行わない。

【0285】 f) タイマを設定してスリープする。

(3.4.3) 緊急処理

システム状態が以下の場合には、ここで述べる処理手順に従いメタライトデーモン 104 は動作する。

◎ログボリュームの残りが非常に少ない。

◎メタキャッシュ 130 の残りが非常に少ない。

【0286】 a) トランザクションの新規開始を制限する。

b) メタライトリストの先頭から、メタデータの状態を調べる。

c) メタデータが書き出し可能状態であれば、その状態

を I/O 中状態に変更し、非同期ライトを発行する。

【0287】 d) ログ未反映状態となっているメタデータがあれば、ログライトデーモン 105 を起動する。

e) b__i o d o n e の関数が I/O の成功を確認する。

【0288】 f) b__i o d o n e の関数がメタライトリストから外す。

g) 繰り返す。

h) 現在の資源状態を調査し、不足があれば、それが解消するまで繰り返す。

【0289】 i) トランザクションの受付を開始する。

j) スリープする。

以下、詳細説明である。

【0290】 a) 緊急時には、新規のトランザクションの開始を受け付けない。具体的には、トランザクションに利用するログバッファを与えないことによって、そのトランザクションの BEGIN 宣言時にてスリープさせる。そのために、特定のフラグを設け、BEGIN 宣言時にそのフラグを参照しなければならない。

【0291】 b) ~ g) ここは基本的に平常処理の場合と同じ処理である。すなわち、メタライトリストの先頭から、追い出し不可状態のメタデータは飛ばして、書き出し可能状態のメタデータを順に非同期ライトによって書き出す。

【0292】 ただし、d) だけ異なる。

d) ログボリュームへの書き出しがデーモンによって行われることから、たとえトランザクションが終了していてもログが未反映のためにメタボリュームへの反映を拒否しているメタデータが存在することが考えられる。この場合は、強制的にログライトデーモン 105 を起動して、書き出し可能状態へ移行するように操作する。既にログライトデーモン 105 が動作中であれば、そのまま先へ進む。

【0293】 h) ここで、現在の資源状態を調べる。平常処理の動作契機となる閾値以上の資源が回復していなければ、再度、メタライトリスト内のメタデータ書き出しを試みる。ここで、d) によってログ未反映状態のメタデータが、書き出し可能となったことによって進展することを期待している。複数回、繰り返すことによって、新規トランザクションの受付を制限していることから、資源回復までメタデータの書き出しが可能であると考える。

【0294】 i) a) にて制限していたトランザクションの受付を再開する。

j) タイマを設定して、スリープする。

(4) 獲得解放処理

メタデータの割り当て状況を管理するビットマップの状態として、FREE-dirty と ALLOC-dirty を区分し、FREE-dirty なビットマップからは獲得しないようにする。

【0295】 具体的には以下の通りである。

・マウント時にビットマップを読み込む際に、1つを獲得用と定める。簡単のため、複数読み込むビットマップのうち、一番若いもの（最初に読み込むもの）を最初の獲得用ビットマップとする。

・獲得用ビットマップの複製を作成する。

・獲得時には複製を検索して獲得位置を決定し、本体と複製の両方のビットを操作する。

・解放時には本体のみのビットを操作する。

・複製ビットマップには解放が記録されないため、獲得処理が進むと全てのビットが立ち、そのビットマップでは獲得できないと判断される。その場合、現在メモリ上にあるビットマップのうち、CLEANなもの、またはALLOC-dirtyのビットマップを次の獲得用ビットマップと定義し、その複製を作成する。

・メモリ上のビットマップが全てFREE-dirtyである場合には、どれかを追い出し、新しいビットマップを獲得用ビットマップとして読み込む。そして、その複製を作成する。ここで、追い出し・読み込みのアルゴリズムは従来通りで良い。

【0296】獲得用ビットマップとして選ばれたビットマップは追い出しの対象とはしない。そのためメタキャッシュ域不足によって他から追い出されることはない。また、メタライトリストに繋がったことによって、メタボリュームに反映された場合にも、CLEANな状態にはなるが、複製は更新せず、そのままとする。

【0297】本実施の形態による効果は、以下の通りである。以上説明したように、本発明によれば複数の二次記憶装置に保存されたメタデータのログにボリューム情報を含め、また、有効なログの位置を算出・保存することでリプレイするログ量を減少せしめ、さらに、ログボリューム全体のゼロクリアをする必要をなくすことにより、ログ機構の最大の利点であるところのファイルシステム復元時間の短縮に及ぼす影響を小さくし、オーバーオールコンピュータシステム可用性の増大に寄与するところが大きい。

【0298】さらに、本発明によれば回一トランザクション内で複数回更新されるメタデータのログを一度しか採取せず、また、ログバッファの分割や、獲得・解放処理の特殊なログ採取方式によって複数のトランザクションの独立性を考慮し、かつ、ログ機構導入による速度性能の劣化を最小に留めることにおいて寄与するところが大きい。

【0299】加えて、本発明によればトランザクション毎に異なるログの採取量を考慮し、多くのメタデータを更新するトランザクションについては分割し、トランザクションに与えられたパラメタをもログに採取し、リプレイ時にそのパラメタを元に、途中で終わったトランザクションを再度実行することによって、オペレーションのセマンティクスを保証した復元が可能となる面において寄与するところが大きい。

【0300】なお、上記の処理機能は、コンピュータによって実現することができる。その場合、説明した処理内容は、コンピュータで読み取り可能な記録媒体に記録されたプログラムに記述されており、このプログラムをコンピュータで実行することにより、上記処理がコンピュータで実現される。コンピュータで読み取り可能な記録媒体としては、磁気記録装置や半導体メモリ等がある。市場へ流通させる場合には、CD-ROM(Compact Disk Read Only Memory)やフロッピーディスク等の可搬型記録媒体にプログラムを格納して流通させたり、ネットワークを介して接続されたコンピュータの記憶装置に格納しておき、ネットワークを通じて他のコンピュータに転送することもできる。コンピュータで実行する際には、コンピュータ内のハードディスク装置等にプログラムを格納しておき、メインメモリにロードして実行する。

【0301】

【発明の効果】以上説明したように、第1の発明では、メタキャッシュ内のメタデータとともにメタデータがどのメタボリュームから取り出されたのかを示すメタデータ管理情報をログとして採取するようにしたため、保持されたログがどのメタボリュームのメタデータに関するログであるのかを管理することができる。その結果、複数のログボリュームにメタデータが格納されていても、ファイルシステムの不整合を修正することが可能となる。

【0302】また、第2の発明では、ログデータの有効範囲を管理するようにしたため、ファイルシステムを復元する際には、ログボリューム12内の有効なログのみを用いて効率よく復元処理を行うことが可能となる。

【0303】また、第3の発明では、ログデータに対して付与するシーケンス番号の最大値を、システムの使用可能年数以上使い続けることができる値としたことで、常に昇順の採番が可能となり、ログボリュームのゼロクリアに伴う処理の遅延を避けることができる。

【0304】また、第4の発明では、メタデータが複数回更新される場合には、最終形態のみをログとして採取するようにしたため、ログデータが短縮されるとともに、ログデータの短縮に伴いファイルシステム復元時間の短縮が図れる。

【0305】また、第5の発明では、割り当て管理情報の一部の複製を獲得操作管理情報とし、メタデータの獲得時には獲得操作管理情報内から取得すべきメタデータを特定するが、解放時には獲得操作管理情報の情報を更新しないようにしたため、メタデータの解放直後に別のトランザクションにより獲得されることがなくなる。その結果、システムダウン時に中途までしか終了していなかった解放トランザクションが解放したはずである領域は、解放される直前の状態のまま保全されることが保証される。

【0306】また、第6の発明では、獲得、解放操作のログとして、その操作対象となった情報のみを記録するようにしたため、ログ採取量が少量ですむ。また、第7の発明では、複数のログバッファを設け、さらにいくつか異なるサイズのログバッファを用意し、トランザクション毎のログをそのトランザクションに適したサイズのログバッファに格納するようにしたため、複数のトランザクションが並列実行される際のトランザクションの独立性を高めることができ、また、メモリ空間を有効に活用できる。

【0307】また、第8の発明では、1つのトランザクションのログがログバッファに収まらない場合に、中間ログとして完結されたログをログボリュームに書き出すようにしたため、部分的であるログを、ファイル復元時には1つのトランザクションのログと同値に扱うことが可能となり、ファイルシステムの復元時に望ましい状態に復元するのが容易となる。

【0308】また、第9の発明では、ログ採取に関するシステムの状態によってトランザクションの受け入れを制限するようにしたため、複数のトランザクションが並列実行されることによるメモリ枯渇等の障害の発生を防止することができる。

【図面の簡単な説明】

【図1】第1の発明の原理構成図である。

【図2】第2の発明の原理構成図である。

【図3】第3の発明の原理構成図である。

【図4】第4の発明の原理構成図である。

【図5】第5の発明の原理構成図である。

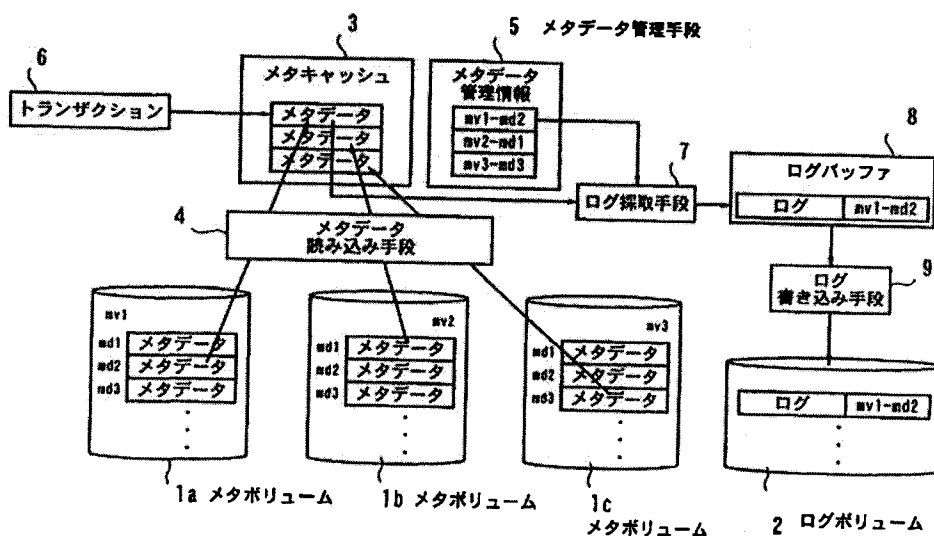
【図6】第6の発明の原理構成図である。

【図7】第7の発明の原理構成図である。

【図8】第8の発明の原理構成図である。

【図9】第9の発明の原理構成図である。

【図1】



【図10】本発明を適用するデータ処理装置のハードウェア構成図である。

【図11】ファイルシステム上で動作するログ採取機能の構成図である。

【図12】メタデータ管理情報を示す図である。

【図13】ログバッファの形式を示す図である。

【図14】有効範囲を説明する図である。

【図15】ログ採取処理のフローチャートである。

【図16】メタボリュームの割り当て管理状況を示す図である。

【図17】ビットマップによるメタデータ獲得処理を示すフローチャートである。

【図18】解放処理のフローチャートである。

【図19】トランザクションの処理の獲得と終了の状況を示す図である。

【図20】ログバッファへのログの格納状況を示す図である。

【図21】ログ採取手順を示すフローチャートである。

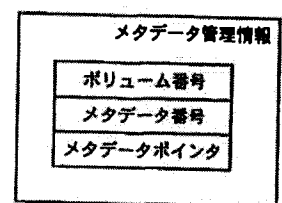
【図22】ファイルシステム復元処理を示すフローチャートである。

【図23】新規トランザクションの受け入れ可否判定処理を示すフローチャートである。

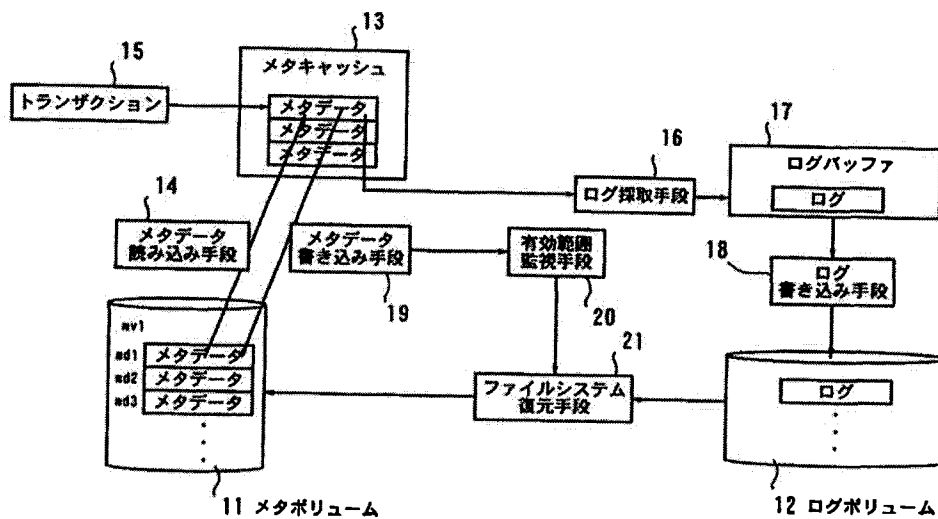
【符号の説明】

- 1 a～1 c メタボリューム
- 2 ログボリューム
- 3 メタキャッシュ
- 4 メタデータ読み込み手段
- 5 メタデータ管理情報
- 6 トランザクション
- 7 ログ採取手段
- 8 ログバッファ
- 9 ログ書き込み手段

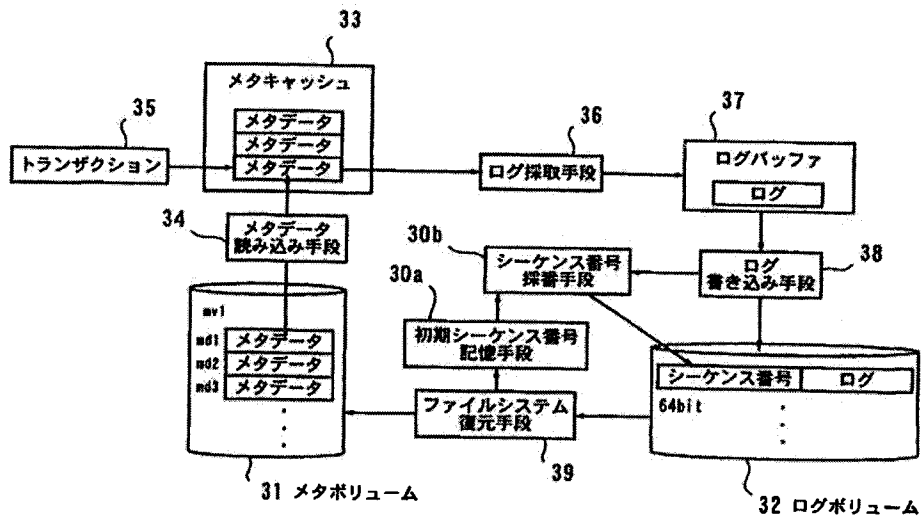
【図12】



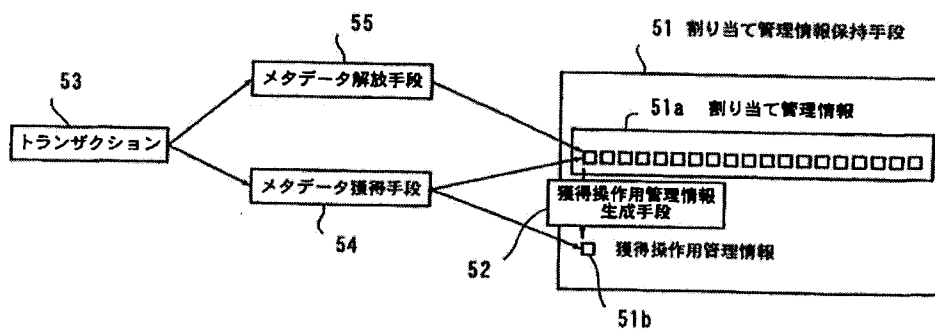
【図 2】



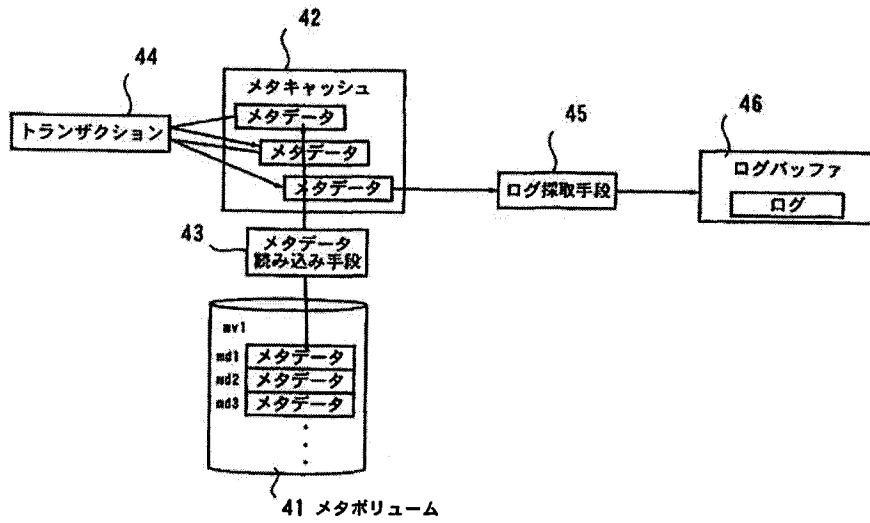
【図 3】



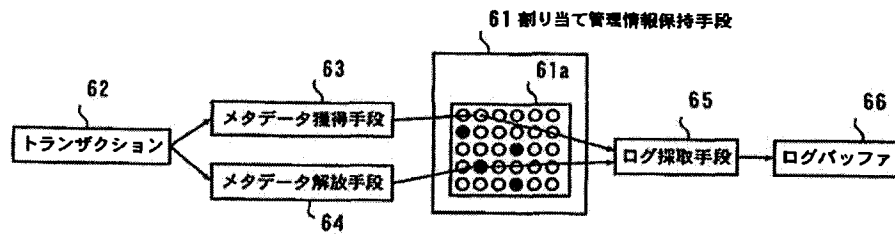
【図 5】



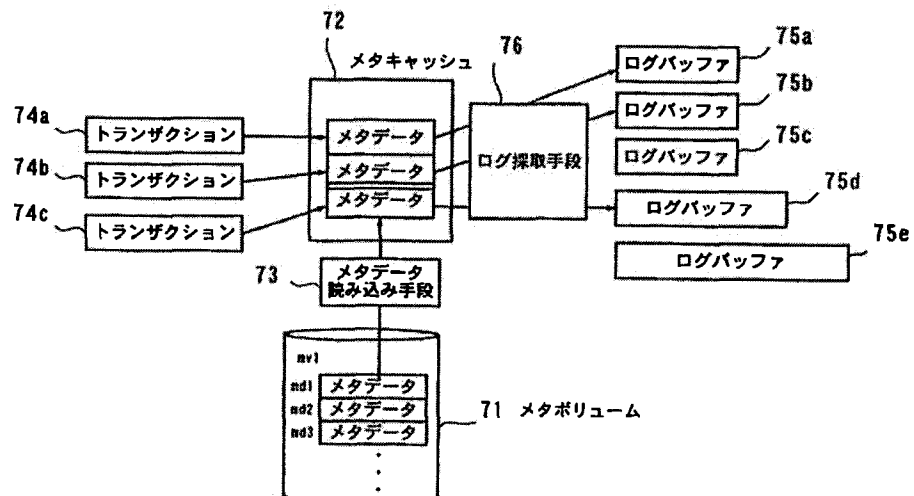
【図 4】



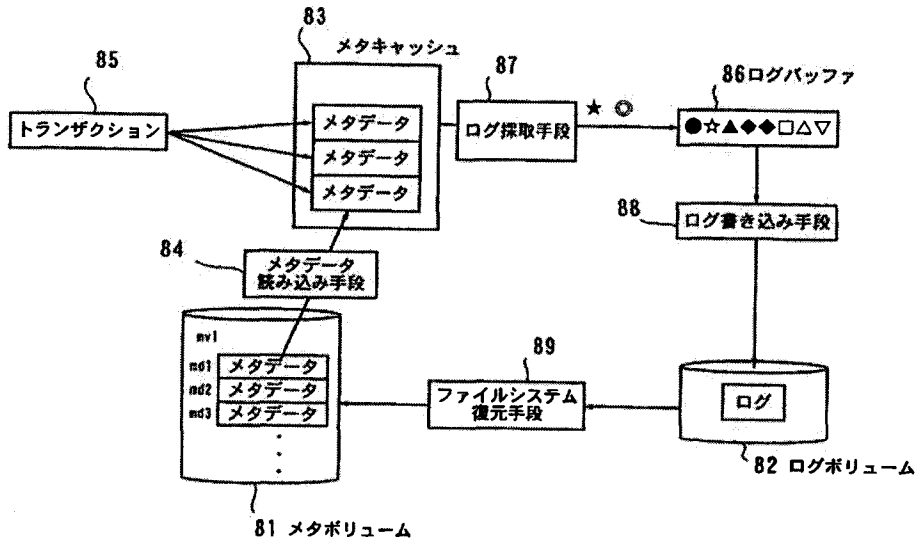
【図 6】



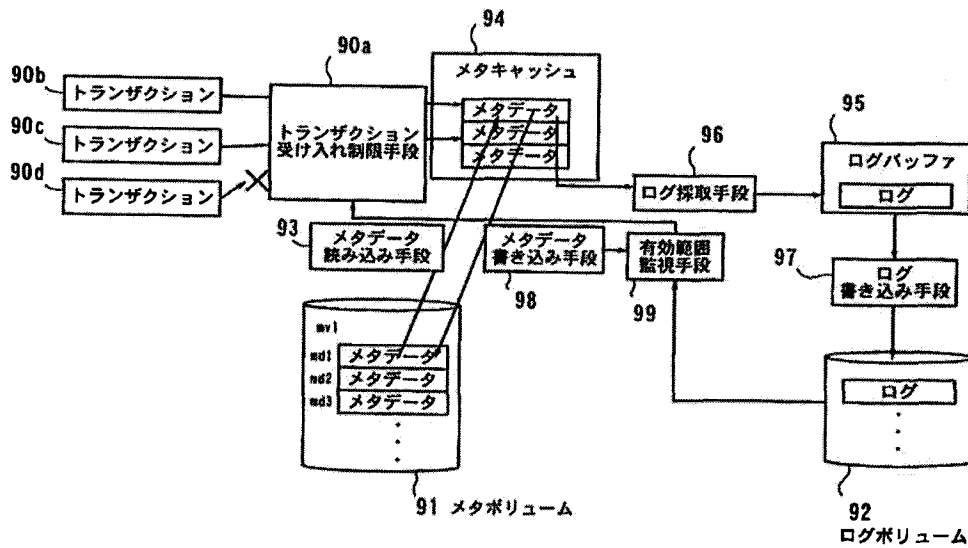
【図 7】



【図8】



【図9】



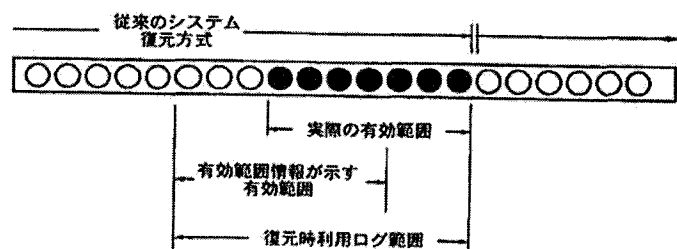
【図13】

ログの形式

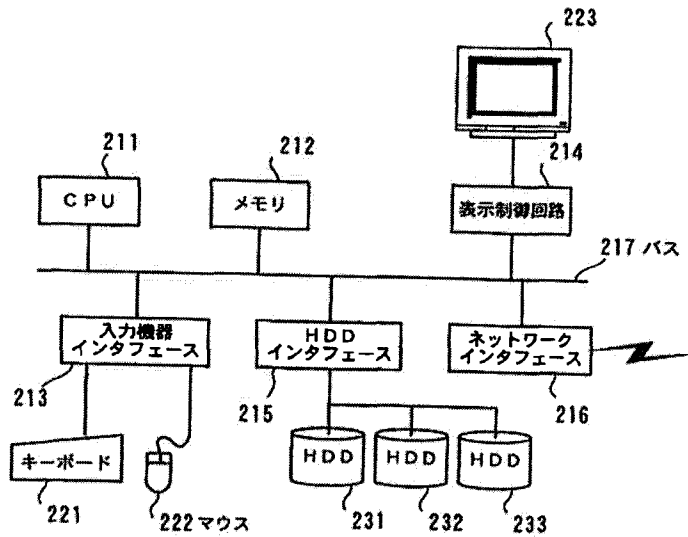
BEGIN マーク	ボリューム 番号	メタデータ 番号	メタデータ 実体	END マーク
--------------	-------------	-------------	-------------	------------

【図14】

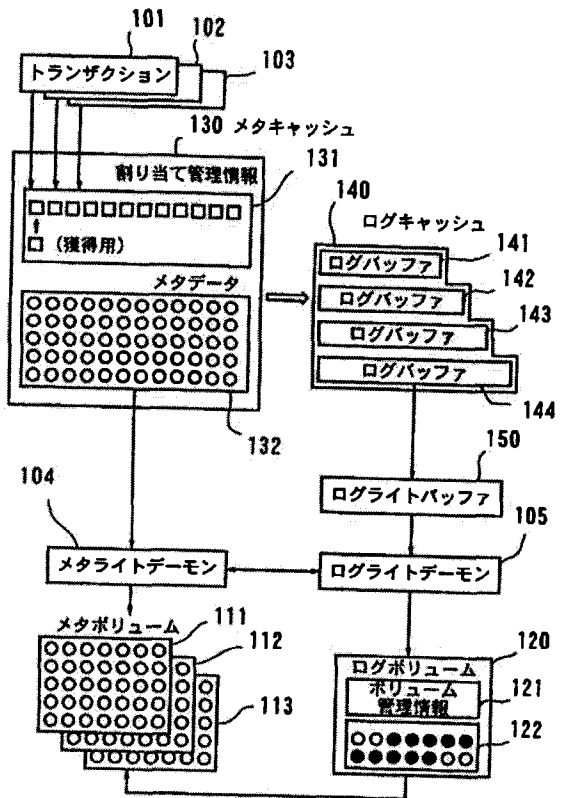
ログボリューム



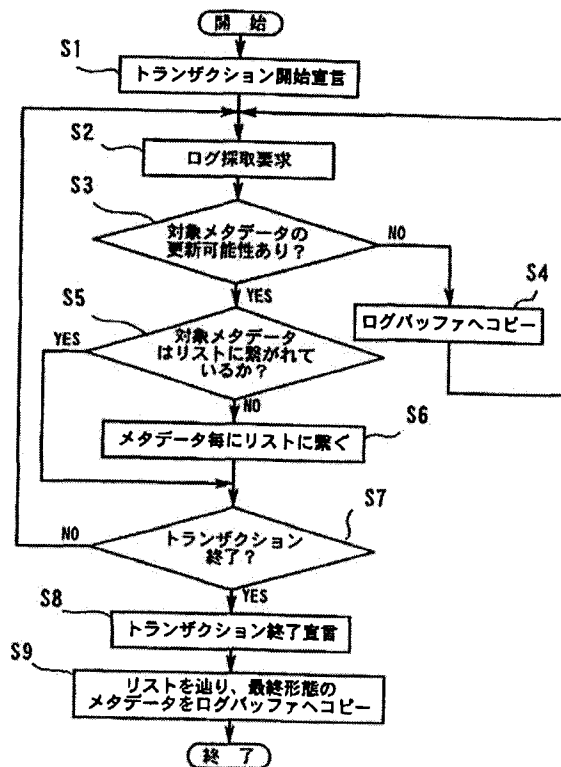
【図10】



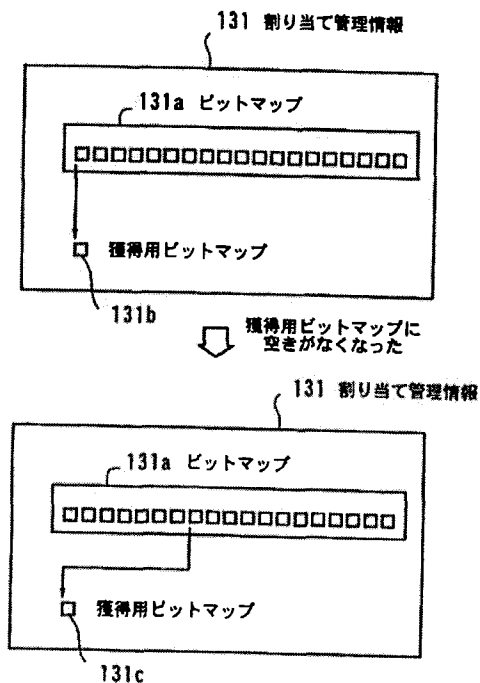
【図11】



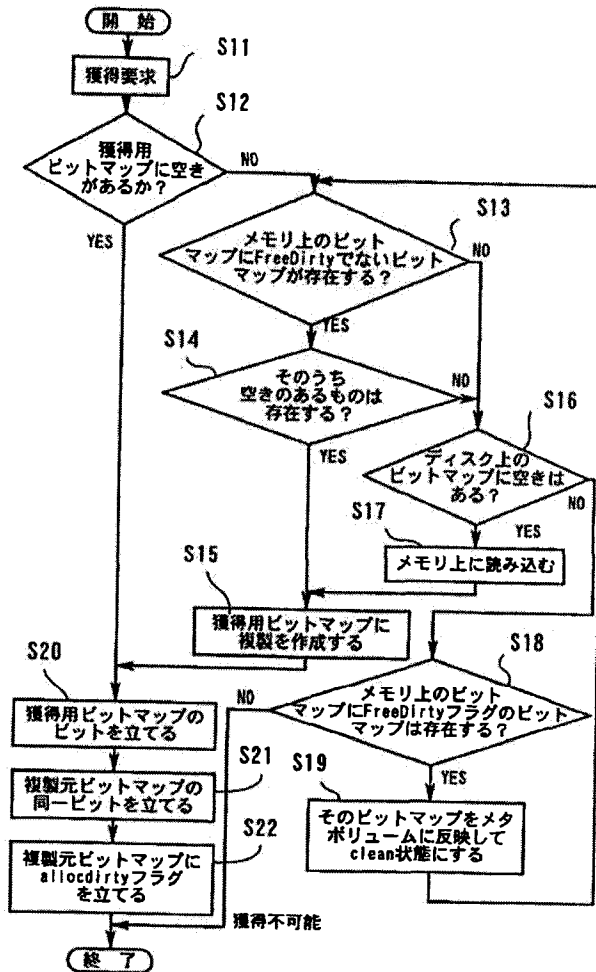
【図15】



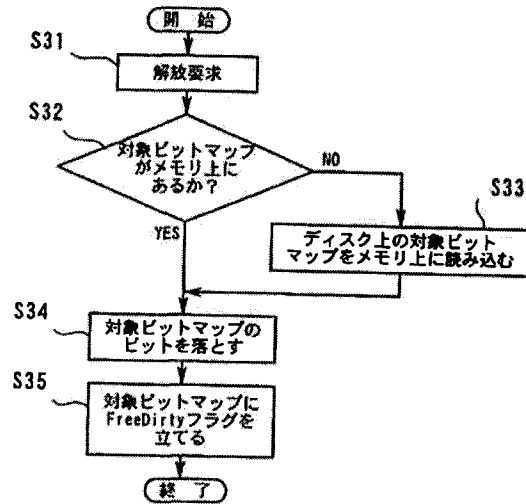
【図16】



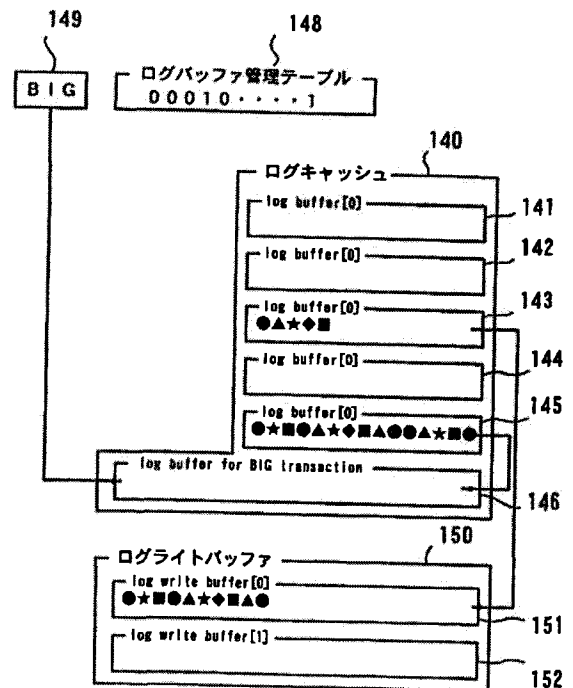
【図17】



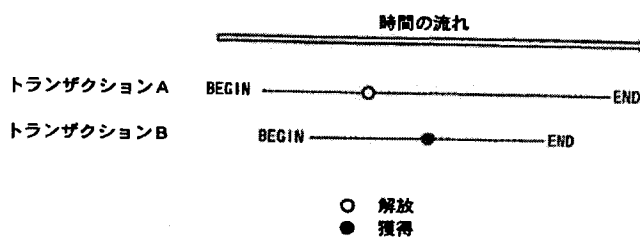
【図18】



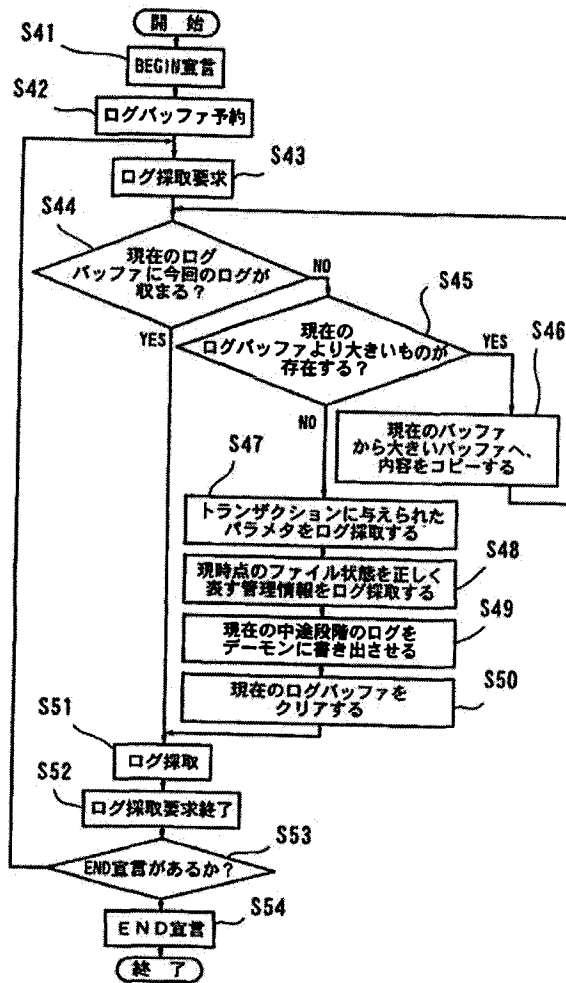
【図20】



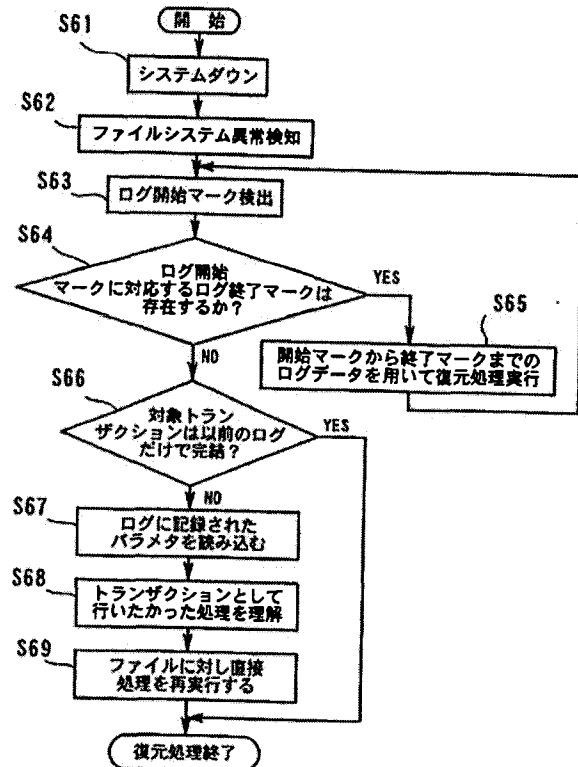
【図19】



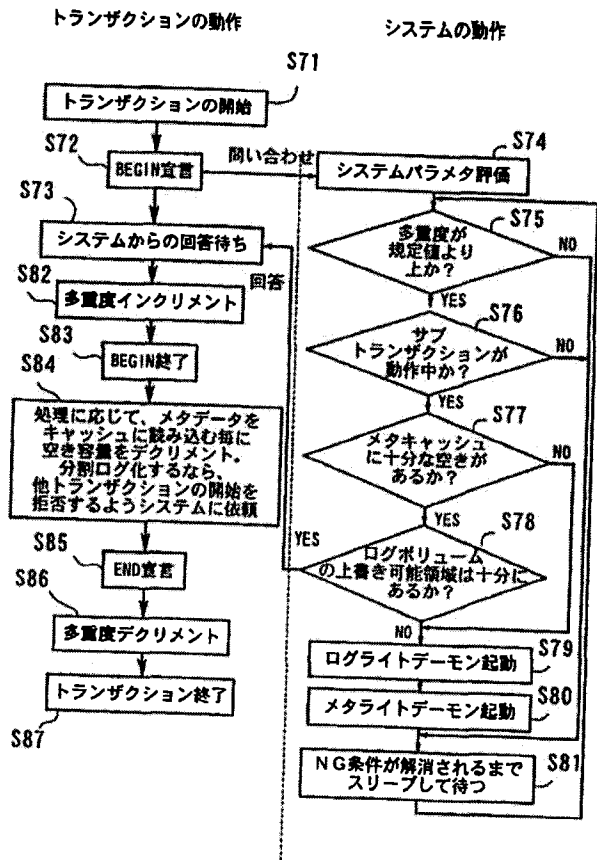
【図 21】



【図 22】



【図 23】



【手続補正書】

【提出日】平成12年2月22日(2000.2.22)

【手続補正1】

【補正対象書類名】明細書

【補正対象項目名】全文

【補正方法】変更

【補正内容】

【書類名】明細書

【発明の名称】データ処理装置及び記録媒体

【特許請求の範囲】

【請求項1】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置である複数のメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュに読み込まれたメタデータが格納されていたメタボリュームの識別情報を管理するメタデータ管理手段と、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、を有することを特徴とするデータ処理装置。

【請求項2】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュ内で変更されたメタデータの内容を

ログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログボリューム内の領域を定期的に循環するようにして、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段と、前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応する前記ログボリューム内のログを、有効なログとして指定する有効範囲監視手段と、ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログの中で、前記有効範囲監視手段により有効なログとして指定されているログのみを用いて、前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、を有することを特徴とするデータ処理装置。

【請求項3】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、前記ログボリュームに格納されたログを用いて前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する初期シーケンス番号記憶手段と、前記ログ書き込み手段がログの書き込みを行う際に、シーケンス番号を昇順で採番し、採番したシーケンス番号を書き込むべきログに付与しており、前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した直後には、前記初期シーケンス番号記憶

手段に格納されたシーケンス番号を基準として採番するシーケンス番号採番手段と、
を有することを特徴とするデータ処理装置。

【請求項4】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、
前記トランザクションの種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、前記メタキャッシュ内で変更されたメタデータの最終形態のみをログとして採取するログ採取手段と、
を有することを特徴とするデータ処理装置。

【請求項5】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
メタデータに対する割り当てを管理するための割り当て管理情報を複数の領域に分割して保持する割り当て管理情報保持手段と、
前記割り当て管理情報保持手段内の割り当て管理情報の一部領域の複製を生成し、獲得操作管理情報とするとともに、前記獲得操作管理情報内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を前記獲得操作管理情報とする獲得操作管理情報生成手段と、
メタデータの獲得及び解放要求を出力するトランザクションと、
前記トランザクションによりメタデータの獲得要求が出された場合には、前記獲得操作管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記獲得操作管理情報と前記割り当て管理情報との内容を変更するメタデータ獲得手段と、
前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、
を有することを特徴とするデータ処理装置。

【請求項6】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
メタデータに対する割り当てを管理するための割り当て管理情報を保持する割り当て管理情報保持手段と、
メタデータの獲得及び解放要求を出力するトランザクションと、
前記トランザクションによりメタデータの獲得要求が出された場合には、前記割り当て管理情報の中の未獲得の

メタデータを獲得し、獲得したメタデータを獲得済みとするように前記割り当て管理情報の内容を変更するメタデータ獲得手段と、

前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、

前記割り当て管理情報内の前記メタデータ獲得手段及び前記メタデータ解放手段によって変更された部分の情報をログとして採取するログ採取手段と、
を有することを特徴とするデータ処理装置。

【請求項7】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新する複数のトランザクションと、
ログをトランザクション毎に保持する、サイズの異なる複数のログバッファと、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて前記ログバッファに格納するログ採取手段と、
を有することを特徴とするデータ処理装置。

【請求項8】 前記ログ採取手段は、最初にログを格納する場合には、トランザクションの内容によって予想される処理に適した大きさの前記ログバッファに格納し、格納対象となる前記ログバッファの記憶容量が不足してきたら、より大きな記憶容量のログバッファへログを移し替え、以後、より大きな記憶容量のログバッファをログの格納対象とすることを特徴とする請求項7記載のデータ処理装置。

【請求項9】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、
ログをトランザクション毎に保持するログバッファと、

前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記ログバッファに格納するログ採取手段と、

前記トランザクションが終了した場合に前記ログバッファの内容を前記ログボリュームに書き込むとともに、前記トランザクションによるログが前記ログバッファ内に格納しきれない場合には、前記ログバッファ内のデータを完結したログに加工し、中間ログとして前記ログボリューム内に格納するログ書き込み手段と、
を有することを特徴とするデータ処理装置。

【請求項10】 前記ログ書き込み手段は、前記中間ログに対して、トランザクションを実行するのに必要とされたパラメタに関する情報を付加し、
ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログを用いて、前記メタボリューム内のメタデータの不整合を修正するとともに、前記中間ログを発見すると、前記中間ログに含まれたパラメタを用いて、前記トランザクションを再実行させるファイルシステム復元手段をさらに有することを特徴とする請求項9記載のデータ処理装置。

【請求項11】 ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新する、複数同時実行可能なトランザクションと、
前記トランザクションからの開始要求を受け付けると、ログ採取に関するシステムの動作状況を判断し、前記トランザクションの受け入れ可否を判断するトランザクション受け入れ判断手段と、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、
前記ログ採取手段が採取した情報を保持するログバッファと、
前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、
を有することを特徴とするデータ処理装置。

【請求項12】 前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段と、
前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応する前記ログボリューム内のロ

グを、有効なログとして指定する有効範囲監視手段とをさらに有し、

前記トランザクション受け入れ判断手段は、前記有効範囲監視手段によって有効なログとされたログがログボリューム中に占める割合が一定値以上である間は、前記トランザクションの受け入れを拒絶することを特徴とする請求項11記載のデータ処理装置。

【請求項13】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
ファイルを管理するメタデータを記憶するための二次記憶装置である複数のメタボリューム、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
前記メタキャッシュに読み込まれたメタデータが格納されていたメタボリュームの識別情報を管理するメタデータ管理手段、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取するログ採取手段、
前記ログ採取手段が採取した情報を保持するログバッファ、
前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項14】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
前記メタキャッシュ内で変更されたメタデータの内容を

ログとして採取するログ採取手段、
前記ログ採取手段が採取した情報を保持するログバッファ、
前記ログボリューム内の領域を定期的に循環するようにして、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段、
前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段、
前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応する前記ログボリューム内のログを、有効なログとして指定する有効範囲監視手段、
ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログの中で、前記有効範囲監視手段により有効なログとして指定されているログのみを用いて、前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項15】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、
メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段、
前記ログ採取手段が採取した情報を保持するログバッファ、
前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段、
前記ログボリュームに格納されたログを用いて前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段、
前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する初期シーケンス番号記憶手段、
前記ログ書き込み手段がログの書き込みを行う際に、システムの使用可能年数以上使い続けることができる値を最大値としたシーケンス番号を昇順で採番し、書き込む

べきログに付与しており、前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した直後には、前記初期シーケンス番号記憶手段に格納されたシーケンス番号から昇順で採番するシーケンス番号採番手段、

としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項16】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、
メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、
メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、
前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、
前記トランザクションの種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、前記メタキャッシュ内で変更されたメタデータの最終形態のみをログとして採取するログ採取手段、
としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項17】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、
メタデータに対する割り当てを管理するための割り当て管理情報を複数の領域に分割して保持する割り当て管理情報保持手段、
前記割り当て管理情報保持手段内の割り当て管理情報の一部領域の複製を生成し、獲得操作用管理情報とするとともに、前記獲得操作用管理情報内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を前記獲得操作用管理情報とする獲得操作用管理情報生成手段、
メタデータの獲得及び解放要求を出力するトランザクション、
前記トランザクションによりメタデータの獲得要求が出された場合には、前記獲得操作用管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記獲得操作用管理情報と前記割り当て管理情報との内容を変更するメタデータ獲得手段、
前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段、

としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項18】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、メタデータに対する割り当てを管理するための割り当て管理情報を保持する割り当て管理情報保持手段、メタデータの獲得及び解放要求を出力するトランザクション、前記トランザクションによりメタデータの獲得要求が出された場合には、前記割り当て管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記割り当て管理情報の内容を変更するメタデータ獲得手段、前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段、前記割り当て管理情報内の前記メタデータ獲得手段及び前記メタデータ解放手段によって変更された部分の情報をログとして採取するログ採取手段、としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項19】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、前記メタキャッシュ内に読み込まれたメタデータの内容を更新する複数のトランザクション、ログをトランザクション毎に保持する、サイズの異なる複数のログバッファ、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて前記ログバッファに格納するログ採取手段、としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項20】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、

メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュ、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクション、ログをトランザクション毎に保持するログバッファ、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記ログバッファに格納するログ採取手段、前記トランザクションが終了した場合に前記ログバッファの内容を前記ログボリュームに書き込むとともに、前記トランザクションによるログが前記ログバッファ内に格納しきれない場合には、前記ログバッファ内のデータを完結したログに加工し、中間ログとして前記ログボリューム内に格納するログ書き込み手段、としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項21】 ログを用いてファイルシステムの不整合の修正を行うファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリューム、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリューム、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段、前記メタキャッシュ内に読み込まれたメタデータの内容を更新する、複数同時実行可能なトランザクション、前記トランザクションからの開始要求を受け付けると、ログ採取に関するシステムの動作状況を判断し、前記トランザクションの受け入れ可否を判断するトランザクション受け入れ判断手段、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段、前記ログ採取手段が採取した情報を保持するログバッファ、前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段、としてコンピュータを機能させることを特徴とするファイル管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明はファイルシステムの修正機能を有するデータ処理装置及び記録媒体に関し、特にログを用いてファイルシステムの不整合の修正を行うデータ処理装置及び記録媒体に関する。

【0002】

【従来の技術】コンピュータシステムを運用していると、何らかの理由によりシステムがダウンする場合がある。突然のシステムダウンが発生すると、ファイルシステムの不整合が生じる。そこで、システムダウン後のブート時には、従来であればファイルシステム全体を走査し、その矛盾点の検出を行う。矛盾点が発見されたら、場合に依じた変更をファイルシステムに加えることによって、ファイルシステムの整合性を回復していた。ところが、ファイルシステム全体を走査するには多くの時間が必要であり、結果としてシステムダウン後の復旧の遅れを招いていた。

【0003】そこで近代のUNIXオペレーティングシステム(OS)のような多くのコンピュータOSのファイルシステムでは、ファイルシステムオペレーションにおいてファイルシステムに保存されているデータが更新される度に、その更新情報をログ(ジャーナル)として採取している。ファイルシステムオペレーション時に更新情報のログ採取を行っておくことにより、システムダウン後のブート時のファイルシステム整合性チェックのフェーズでは、残されたログを順次走査し、対応する領域へアップデートすることによって、ファイルシステムの整合性が保証される。その結果、システムのダウン時間が短縮される。

【0004】

【発明が解決しようとする課題】しかし、ログ機構を導入することにより、次に挙げる問題が生じていた。

(1) 第1の問題点

ファイルを管理するメタデータ(二次記憶装置に格納されたファイルを管理するためのデータ)は、ファイルシステムオペレーション時に二次記憶装置からメモリへ読み込まれ、メモリ内において操作される。その後、所定のタイミングで二次記憶装置へとその更新内容が反映される。ログ機構の導入時には、その二次記憶装置への反映前に更新内容をログ専用の二次記憶装置へと記録する必要がある。

【0005】しかし、大規模ファイルシステムに対応するために、複数の二次記憶装置がメタデータに割り当てられ、異なる二次記憶装置に割り当てられたメタデータを1つのファイルシステムオペレーションが操作する場合がある。このファイルシステムオペレーションのログを採取する際に、メモリ内のメタデータだけをログとして記録していたのでは、残されたログからメタデータ毎に異なる二次記憶装置を検索するのに時間を消費し、ログ機構導入による最大のメリットである、ファイルシ

テム復元の時間短縮に悪い影響を与える。

【0006】(2) 第2の問題点

同様にファイルシステム復元の時間短縮に悪い影響を与える要因として、有限なサイズしかないログ専用の二次記憶装置全体をファイルシステム復元時に探索することが考えられる。

【0007】ログ機構の導入はファイルシステムオペレーションを細分化したトランザクションの完了を常に保証しつつ動作するため、ログ機構を導入した多くのファイルシステムはトランザクションにシーケンス番号を与え、ファイルシステム復元時にはシーケンス番号をもとに最古のトランザクションを同定する。そして、最古のトランザクションからログリプレイと呼ばれるログを用いたファイルシステム復元操作を開始する。

【0008】ここで、有限サイズのログ専用の二次記憶装置内には、ファイルシステムの整合性を復元するために欠かせないログが確かに存在する可能性があるが、有限なサイズを有効利用するためにログ専用の二次記憶装置は過去のログを上書きしてサイクリックに利用しなければならない。このような処理を行うには、ある程度以上古いログが必ず不必要となっていることが前提となる。従って、多くのログの内容は既に不必要となっている。すなわち、ログ専用の二次記憶装置全体を探索あるいは反映するのは非効率的である。

【0009】(3) 第3の問題点

最古のトランザクションを特定するためのシーケンス番号は単調増加であることが要求され、運用途中にオーバーフローによってゼロに戻されてしまうことは許されない。これを回避するために、ファイルシステム復元作業終了時、またはオーバーフロー直前にログ専用の二次記憶装置全体をゼロクリアし、再度シーケンス番号ゼロから順にトランザクションを処理するのが一般的である。しかし、ログ専用の二次記憶装置をゼロクリアするためには多くの時間が必要となる。少なくともシステムの運用を一時停止する必要がある、サーバ装置などによるサービスの提供を停止せざるを得なくなってしまう。

【0010】以上の(1)～(3)の課題はファイルシステム復元時の問題であるが、ログ機構を導入することは通常の運用時にも問題を引き起こす可能性を持っている。ログ機構は、メモリ上にログをスプールする手法や、ログ専用の二次記憶装置として用いられるディスクの特性を考慮したシーケンシャルアクセスなど、高速化のための条件は整えられているが、ログの採取の仕方を熟考しなければ、ログ採取に伴う性能劣化は非常に大きいものとなりうる。

【0011】(4) 第4の問題点

単一のトランザクション内で、同一データを複数回更新することは度々あるが、その都度、その同一データに対するログを採取したのではメモリが不足し、二次記憶装置へのI/O量が増加する。

【0012】(5)第5の問題点

ログ機構の導入はトランザクションの順序性を保証し、終了したトランザクションのログを順に採取することを要求するために、あるトランザクションが操作したログ対象データを他のトランザクションが操作することが一般的に不可能な状況となりうる。ここで、個々のファイルの内容を対象とするトランザクションについては、ファイル単位に排他制御を行うことによって、複数のトランザクションが並列実行することは比較的容易である。しかしながら、個々のファイルによらないもの、特に領域の割り当て情報を操作する場合には、並列実行がきわめて難しくなる。

【0013】領域の獲得・解放処理が並列に動作する場合を考えると、それぞれが獲得、解放のログを採取する。同一の管理情報（ここでは、ビットマップを考える）によって管理される領域の獲得・解放処理であった場合には、同一の管理情報が、解放された状態、獲得された状態でログに記録されることになる。

【0014】ファイルシステムの復元処理を行わなければならないようなシステムダウンが発生するタイミングによっては、このように別トランザクションの更新情報を含むログの採取方式では様々な問題が生じる。

【0015】Aというトランザクションが解放処理、Bというトランザクションが獲得処理を行う場合を考える。ここで、トランザクションAの解放処理はトランザクションBの獲得処理よりも先に行われ、かつ、トランザクションAはトランザクションBよりも後に終わる場合を例に挙げる。

【0016】このとき、トランザクションAが解放した領域をトランザクションBが獲得してしまう場合が考えられる。トランザクションBが先に終了することから、残されたログには、まず獲得処理が記録され、次に解放処理が記録される。

【0017】上記の場合のトランザクションBのログだけが記録されており、それを復元に用いた場合には、本来行われたはずである解放処理の記録が残されていないことから、該当領域が二重に獲得された状態になってしまう。トランザクションAのログまで記録されており、復元に用いられると、トランザクションBが利用している領域がトランザクションAの解放処理のログによって解放されている状態になってしまう。いずれも本来の状態とは異なっており、避けなければならない。しかしながら、これを回避するために並列実行を制限することは、マルチタスクを実現したOS上のファイルシステムの速度性能の低下に与える影響が非常に大きい。

【0018】(6)第6の問題点

既に述べたようにログ機構の第一の目的としてファイルシステムを復元するために費やす時間の短縮が挙げられるが、そのためにトランザクションの独立性を疎かにしたり、中途半端な状態での整合性回復によりあたかも正

常に動作しているかのように振る舞うファイルシステムが多く見受けられる。

【0019】従来のメタデータ管理方式のように、ログを記録するためのメモリ空間をファイルシステム全体で1つのキャッシュメモリを共有していたのではトランザクション毎の独立性を保つのが難しく、他のトランザクションによる更新情報が1つのトランザクションのログとして記録されてしまう可能性が高い。特にファイルに対する排他で制御しきれない割り当て管理情報については前述の通りである。

【0020】(7)第7の問題点

トランザクション毎に必要なログのサイズが異なることから、複数のログバッファとしてログを記録するためのメモリをトランザクション毎に割り当て、管理する場合に、全てのメモリサイズが同一である必要はない。例えば、ファイルの参照時刻を更新するだけのトランザクションが残すログのサイズは大変小さく、巨大なデータ書き込み要求のトランザクションは必然的にそれだけ大きいログサイズとなる。

【0021】(8)第8の問題点

トランザクション毎に残すログサイズの違いを考慮して、限られたメモリ空間の有効利用を試みても、キャッシュメモリサイズは残されるログサイズに比較して、やはり小さい。

【0022】(9)第9の問題点

「第8の問題点」の解決策として、1つのトランザクションを分割し、中途の状態のログを出力することが考えられる。しかし、一般にファイルを管理するメタデータは、トランザクションが中途の状態ではやはり中途の状態であり、そのままログに記録したところで、そのログを用いて復元されるファイルシステムは中途の状態にかなり得ない。これではオペレーションのセマンティクスを保証した復元とはなり得ない。

【0023】(10)第10の問題点

「第9の問題点」で示した中途の状態でのトランザクションの中断はファイルシステムにとって危険な状態といえる。ログ機構の導入は、採取したログをログボリュームに反映するまでは該当メタデータもキャッシュメモリから追い出せないことを意味する。ログがキャッシュメモリに入りきらないほど巨大となっているときには、メタデータを管理するキャッシュメモリの利用率も高くなっていることが考えられる。ログを出力するまでメタデータをキャッシュメモリから追い出せないため、このような状態で多くのトランザクションが並列実行されると、メモリ枯渇によるハングアップ状態に陥ることも考えられる。

【0024】(11)第11の問題点

第10の問題点と同様の資源枯渇はログ記録用の二次記憶装置についても言える。キャッシュメモリに比較すれば巨大な二次記憶装置についても、メタデータ記録用の

二次記憶装置へのI/Oを削減するために、多くのログを有効なログとして残しておけば、それは利用可能な領域の減少を引き起こす。その際に多数のトランザクションの並列実行を許すことはメタデータキャッシュ枯渇、ログキャッシュ枯渇、ログ記録用二次記憶装置枯渇、などを誘発する可能性がある。

【0025】本発明はこのような点に鑑みてなされたものであり、ファイルシステム修復処理の効率化を図ったデータ処理装置を提供することを目的とする。

【0026】

【課題を解決するための手段】本発明では上記課題を解決するための第1の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置である複数のメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュに読み込まれたメタデータが格納されていたメタボリュームの識別情報を管理するメタデータ管理手段と、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、を有することを特徴とするデータ処理装置が提供される。

【0027】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。その際、読み込んだメタデータがどのメタボリュームから読み込まれたものであるのかが、メタデータ管理手段によって管理される。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。この際、メタデータが格納されていたメタボリュームの識別情報をも採取する。採取した情報は、ログバッファに保持される。そして、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。

【0028】また、上記課題を解決する第2の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するため

の二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログボリューム内の領域を定期的に循環するようにして、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、前記メタキャッシュ内のメタデータをメタボリューム内に格納するメタデータ書き込み手段と、前記メタデータ書き込み手段による書き込み動作を監視しており、変更内容が前記メタボリューム内に反映されていないメタデータに対応する前記ログボリューム内のログを、有効なログとして指定する有効範囲監視手段と、ファイルシステム復元要求を受け取ると、前記ログボリュームに格納されたログの中で、前記有効範囲監視手段により有効なログとして指定されているログのみを用いて、前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、を有することを特徴とするデータ処理装置が提供される。

【0029】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファに保持される。そして、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。その後、メタデータ書き込み手段により、メタキャッシュ内のメタデータがメタボリューム内に格納される。その書き込み動作は、有効範囲監視手段で監視されており、変更内容がメタボリューム内に反映されていないメタデータに対応するログボリューム内のログが、有効なログとして指定される。そして、ファイルシステム復元要求が出されると、ファイルシステム復元手段により、ログボリュームに格納されたログの中で、有効範囲監視手段により有効なログとして指定されているログのみを用いて、メタボリューム内のメタデータの不整合が修正される。

【0030】また、上記課題を解決する第3の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となる

メタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を前記ログボリュームに格納するログ書き込み手段と、前記ログボリュームに格納されたログを用いて前記メタボリューム内のメタデータの不整合を修正するファイルシステム復元手段と、前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する初期シーケンス番号記憶手段と、前記ログ書き込み手段がログの書き込みを行う際に、シーケンス番号を昇順で採番し、採番したシーケンス番号を書き込むべきログに付与しており、前記ファイルシステム復元手段が前記メタボリューム内のメタデータの不整合を修正した直後には、前記初期シーケンス番号記憶手段に格納されたシーケンス番号を基準として採番するシーケンス番号採番手段と、を有することを特徴とするデータ処理装置が提供される。

【0031】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファに保持される。そして、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。その際、シーケンス番号採番手段により、シーケンス番号が昇順で採番され、書き込むべきログに付与される。ファイルシステムに不整合が発生すると、ファイルシステム復元手段により、ログボリュームに残されたログを用いてメタデータの不整合が修正される。このとき、修正に用いられた最後のログのシーケンス番号が初期シーケンス番号記憶手段に記憶される。その後、ログ書き込み手段によりログバッファ内の情報がログボリュームに書き込まれると、シーケンス番号採番手段により、初期シーケンス番号記憶手段に格納されたシーケンス番号を基準としてシーケンス番号が採番され、書き込むべきログに付与される。

【0032】また、上記課題を解決する第4の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッ

シュ内に読み込まれたメタデータの内容を更新するトランザクションと、前記トランザクションの種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、前記メタキャッシュ内で変更されたメタデータの最終形態のみをログとして採取するログ採取手段と、を有することを特徴とするデータ処理装置が提供される。

【0033】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段により、トランザクションの種別が判断され、メタデータの更新を複数回行う可能性のあるトランザクションの場合には、メタキャッシュ内で変更されたメタデータの最終形態のみがログとして採取される。

【0034】また、上記課題を解決する第5の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、メタデータに対する割り当てを管理するための割り当て管理情報を複数の領域に分割して保持する割り当て管理情報保持手段と、前記割り当て管理情報保持手段内の割り当て管理情報の一部領域の複製を生成し、獲得操作管理情報とするとともに、前記獲得操作管理情報内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を前記獲得操作管理情報とする獲得操作管理情報生成手段と、メタデータの獲得及び解放要求を出力するトランザクションと、前記トランザクションによりメタデータの獲得要求が出された場合には、前記獲得操作管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記獲得操作管理情報と前記割り当て管理情報との内容を変更するメタデータ獲得手段と、前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、を有することを特徴とするデータ処理装置が提供される。

【0035】このようなデータ処理装置によれば、獲得操作管理情報生成手段により、割り当て管理情報の一部領域の複製が生成され、獲得操作管理情報とされる。トランザクションにより獲得要求があると、メタデータ獲得手段により、獲得操作管理情報内の未獲得のメタデータが獲得される。すると、獲得操作管理情報と割り当て管理情報との内容が更新される。また、トランザクションよりメタデータの解放要求があると、メタデータ解放手段により該当するメタデータの解放処理が行われる。この際、割り当て管理情報の内容のみが更新される。ここで、獲得操作管理情報内に未獲得のメタデータがなくなると、割り当て管理情報内の別の領域の

複製が獲得操作管理情報とされる。

【0036】また、上記課題を解決する第6の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、メタデータに対する割り当てを管理するための割り当て管理情報を保持する割り当て管理情報保持手段と、メタデータの獲得及び解放要求を出力するトランザクションと、前記トランザクションによりメタデータの獲得要求が出された場合には、前記割り当て管理情報保持手段の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように前記割り当て管理情報の内容を変更するメタデータ獲得手段と、前記トランザクションによりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、前記割り当て管理情報の内容を変更するメタデータ解放手段と、前記割り当て管理情報内の前記メタデータ獲得手段及び前記メタデータ解放手段によって変更された部分の情報をログとして採取するログ採取手段と、を有することを特徴とするデータ処理装置が提供される。

【0037】このようなデータ処理装置によれば、トランザクションからメタデータの獲得要求が出されると、メタデータ獲得手段によって未獲得のメタデータの1つが割り当て管理情報内から獲得され、そのメタデータが獲得済みの状態とされる。また、トランザクションからメタデータの解放要求が出されると、メタデータ解放手段によって該当するメタデータが未獲得の状態に変更される。そして、ログ採取手段により、割り当て管理情報内のメタデータ獲得手段及びメタデータ解放手段によって変更された部分の情報がログとして採取される。

【0038】また、上記課題を解決する第7の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新する複数のトランザクションと、ログをトランザクション毎に保持する、サイズの異なる複数のログバッファと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて前記ログバッファに格納するログ採取手段と、を有することを特徴とするデータ処理装置が提供される。

【0039】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段により、メタキャッシュ内で変更

されたメタデータがログとして採取され、トランザクション毎のログバッファに保持される。

【0040】また、上記課題を解決する第8の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新するトランザクションと、ログをトランザクション毎に保持するログバッファと、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取し、前記ログバッファに格納するログ採取手段と、前記トランザクションが終了した場合に前記ログバッファの内容を前記ログボリュームに書き込むとともに、前記トランザクションによるログが前記ログバッファ内に格納しきれない場合には、前記ログバッファ内のデータを完結したログに加工し、中間ログとして前記ログボリューム内に格納するログ書き込み手段と、を有することを特徴とするデータ処理装置が提供される。

【0041】このようなデータ処理装置によれば、トランザクションがメタデータの更新をする際には、まず、メタデータ読み込み手段によりメタボリューム内の必要なメタデータがメタキャッシュに読み込まれる。ここで、トランザクションがメタデータの内容を更新すると、ログ採取手段によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファに保持される。そして、ログバッファに格納しきれなくなると、トランザクションが終了すると、ログ書き込み手段により、ログバッファ内の情報がログボリュームに書き込まれる。ログバッファに格納しきれなくなった場合には、ログバッファの内容を完結したログに加工され、中間ログとしてログボリュームに格納される。

【0042】また、上記課題を解決する第9の発明として、ログを用いてファイルシステムの不整合の修正を行うデータ処理装置において、ファイルを管理するメタデータを記憶するための二次記憶装置であるメタボリュームと、メタデータの更新結果であるログを記憶するための二次記憶装置であるログボリュームと、メタデータを記憶するために主記憶装置内に設けられたメタキャッシュと、メタデータの内容が変更される際に、対象となるメタデータを前記メタキャッシュへと読み込むメタデータ読み込み手段と、前記メタキャッシュ内に読み込まれたメタデータの内容を更新する、複数同時実行可能なトランザクションと、前記トランザクションからの開始要求を受け付けると、ログ採取に関するシステムの動作状況を判断し、前記トランザクションの受け入れ可否を判

断するトランザクション受け入れ判断手段と、前記メタキャッシュ内で変更されたメタデータの内容をログとして採取するログ採取手段と、前記ログ採取手段が採取した情報を保持するログバッファと、前記ログバッファが保持する情報を、適宜前記ログボリュームに格納するログ書き込み手段と、を有することを特徴とするデータ処理装置が提供される。

【0043】このようなデータ処理装置によれば、トランザクションから開始要求が出されると、トランザクション受け入れ制限手段が受け入れの可否を判断する。その後、メタデータ書き込み手段によるメタデータの書き込みが進み、有効なログの割合が減少したら、その時点でトランザクションの開始要求を許可する。

【0044】

【発明の実施の形態】以下、本発明の実施の形態を図面を参照して説明する。図1は、第1の発明の原理構成図である。この第1の発明は、複数の二次記憶装置がメタデータに割り当てられている場合におけるファイルシステム不整合修復時間の短縮を図るものである。

【0045】このデータ処理装置には、二次記憶装置としてメタボリューム1a、1b、1cとログボリューム2とが設けられている。各メタボリューム1a、1b、1cには、ファイルを管理するためのメタデータが記憶されている。また、ログボリューム2には、メタデータの更新結果であるログが記憶されている。また、主記憶装置内にはメタキャッシュ3が設けられている。メタキャッシュ3は、メタデータを記憶するための主記憶装置内の記憶領域である。メタデータ読み込み手段4は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ3へと読み込む。メタデータ管理手段5は、メタキャッシュ3に読み込まれたメタデータが格納されていたメタボリューム1a、1b、1cの識別情報を管理する。トランザクション6は、メタキャッシュ3内に読み込まれたメタデータの内容を更新する。ログ採取手段7は、メタキャッシュ3内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリューム1a、1b、1cの識別情報を採取する。ログバッファ8は、ログ採取手段7が採取した情報を保持する。ログ書き込み手段9は、ログバッファ8が保持する情報を、適宜ログボリューム2に格納する。

【0046】このようなデータ処理装置によれば、トランザクション6がメタデータの更新をする際には、まず、メタデータ読み込み手段4によりメタボリューム1a～1c内の必要なメタデータがメタキャッシュ3に読み込まれる。その際、読み込んだメタデータがどのメタボリュームから読み込まれたものであるのかが、メタデータ管理手段5によって管理される。ここで、トランザクション6がメタデータの内容を更新すると、ログ採取手段7によって更新後のメタデータがログとして採取さ

れる。この際、メタデータが格納されていたメタボリューム1a～1cの識別情報をも採取する。採取した情報は、ログバッファ8に保持される。そして、ログ書き込み手段9により、ログバッファ8内の情報がログボリューム2に書き込まれる。

【0047】これにより、ログボリューム2に保持されたログがどのメタボリューム1a～1cのメタデータに関するログであるのかを管理することができる。その結果、複数のメタボリューム1a～1cにメタデータが格納されていても、ファイルシステムの不整合を修正することが可能となる。

【0048】図2は、第2の発明の原理構成図である。第2の発明は、ログの有効範囲を監視することで、ファイルシステム復元時の効率化を図ったものである。第2の発明は、以下のような要素で構成される。

【0049】メタボリューム11は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム12は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ13は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段14は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ13へと読み込む。トランザクション15は、メタキャッシュ13内に読み込まれたメタデータの内容を更新する。ログ採取手段16は、メタキャッシュ13内で変更されたメタデータの内容をログとして採取するとともに、採取したメタデータが格納されていたメタボリュームの識別情報を採取する。ログバッファ17は、ログ採取手段16が採取した情報を保持する。ログ書き込み手段18は、ログボリューム12内の領域を定期的に循環するようにして、ログバッファ17が保持する情報をログボリューム12に格納する。メタデータ書き込み手段19は、メタキャッシュ13内のメタデータをメタボリューム11内に格納する。有効範囲監視手段20は、メタデータ書き込み手段19による書き込み動作を監視しており、変更内容がメタボリューム11に反映されていないメタデータに対応するログボリューム12内のログを、有効なログとして指定する。ファイルシステム復元手段21は、ファイルシステム復元要求を受け取ると、ログボリューム12に格納されたログの中で、有効範囲監視手段20により有効なログとして指定されているログのみを用いて、メタボリューム11内のメタデータの不整合を修正する。なお、有効範囲監視手段20による有効なログの指定方法としては、例えば、有効範囲を示す情報を不揮発性の記録媒体（ログボリューム12等）に記録することができる。この場合、ファイルシステム復元手段21は、有効範囲が記録された不揮発性の記録媒体内の情報を読みとることで、有効なログと指定されているログを認識できる。

【0050】このようなデータ処理装置によれば、トラ

ンザクション15がメタデータの更新をする際には、まず、メタデータ読み込み手段14によりメタボリューム11内の必要なメタデータがメタキャッシュ13に読み込まれる。ここで、トランザクション15がメタデータの内容を更新すると、ログ採取手段16によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファ17に保持される。そして、ログ書き込み手段18により、ログバッファ17内の情報がログボリューム12に書き込まれる。その後、メタデータ書き込み手段19により、メタキャッシュ13内のメタデータがメタボリューム11内に格納される。その書き込み動作は、有効範囲監視手段20で監視されており、変更内容がメタボリューム11に反映されていないメタデータに対応するログボリューム12内のログが、有効なログとして指定される。そして、ファイルシステム復元要求が出されると、ファイルシステム復元手段21により、ログボリューム12に格納されたログの中で、有効範囲監視手段20により有効なログとして指定されているログを用いて、メタボリューム11内のメタデータの不整合が修正される。

【0051】これにより、ファイルシステムを復元する際には、ログボリューム12内の有効なログのみを用いて効率よく復元処理を行うことが可能となる。図3は、第3の発明の原理構成図である。第3の発明は、ログに付加するシーケンス番号のデータサイズを十分大きなものとし、シーケンス番号を常に昇順で使い続けることができる（ゼロクリアが不要となる）ようにしたものである。第3の実施の形態は、以下のような要素で構成される。

【0052】メタボリューム31は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム32は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ33は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段34は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ33へと読み込む。トランザクション35は、メタキャッシュ33内に読み込まれたメタデータの内容を更新する。ログ採取手段36は、メタキャッシュ33内で変更されたメタデータの内容をログとして採取する。ログバッファ37は、ログ採取手段36が採取した情報を保持する。ログ書き込み手段38は、ログバッファが保持する情報を前記ログボリュームに格納する。ファイルシステム復元手段39は、ログボリューム32に格納されたログを用いてメタボリューム31内のメタデータの不整合を修正する。初期シーケンス番号記憶手段30aは、ファイルシステム復元手段39がメタボリューム31内のメタデータの不整合を修正した時に用いられたログの最後のシーケンス番号を記憶する。シーケンス番号採番手段30bは、ログ書き込み

手段38がログの書き込みを行う際に、シーケンス番号を昇順で採番し、採番したシーケンス番号を書き込むべきログに付与しており、ファイルシステム復元手段39がメタボリューム31内のメタデータの不整合を修正した直後には、初期シーケンス番号記憶手段30aに格納されたシーケンス番号を基準として採番する。

【0053】このようなデータ処理装置によれば、トランザクション35がメタデータの更新をする際には、まず、メタデータ読み込み手段34によりメタボリューム31内の必要なメタデータがメタキャッシュ33に読み込まれる。ここで、トランザクション35がメタデータの内容を更新すると、ログ採取手段36によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファ37に保持される。そして、ログ書き込み手段38により、ログバッファ37内の情報がログボリューム32に書き込まれる。その際、シーケンス番号採番手段30bによりシーケンス番号が昇順で採番され、書き込むべきログに付与される。また、ファイルシステム復元手段39により、ログボリューム32に格納されたログを用いてメタボリューム31内のメタデータの不整合が修正されると、修正した時に用いられたログの最後のシーケンス番号が初期シーケンス番号記憶手段30aに記憶される。その後、ログ書き込み手段38により、ログバッファ37内の情報がログボリューム32に書き込まれると、シーケンス番号採番手段30bにより、初期シーケンス番号記憶手段30aに記憶されたシーケンス番号を基準としてシーケンス番号が昇順で採番され、書き込むべきログに付与される。

【0054】これにより、既にファイルシステムの復元に用いたログと、ファイルシステム復元後に介したトランザクションのログとのシーケンス番号が重ならないことを保証することができる。その結果、システムが使用可能な期間中にログボリュームをゼロクリアする必要がなくなり、ゼロクリアに伴う処理の遅延を避けることができる。

【0055】図4は、第4の発明の原理構成図である。第4の発明は、同じメタデータに対して複数回更新処理が行われる場合に、最終形態のメタデータのみをログとして採取するものである。第4の発明は、以下のような要素で構成される。

【0056】メタボリューム41は、ファイルを管理するメタデータを記憶するための二次記憶装置である。メタキャッシュ42は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段43は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ42へと読み込む。トランザクション44は、メタキャッシュ42内に読み込まれたメタデータの内容を更新する。ログ採取手段45は、トランザクション44の種別を判断し、メタデータの更新を複数回行う可能性のあるトランザクシ

ンの場合には、メタキャッシュ42内で変更されたメタデータの最終形態のみをログとして採取する。ログバッファ46は、ログ採取手段45が採取した情報を保持する。

【0057】このようなデータ処理装置によれば、トランザクション44がメタデータの更新をする際には、まず、メタデータ読み込み手段43によりメタボリューム41内の必要なメタデータがメタキャッシュ42に読み込まれる。ここで、トランザクション44がメタデータの内容を更新する。すると、ログ採取手段45により、トランザクション44の種別が判断され、メタデータの更新を複数回行う可能性のあるメタデータ属性を予測する。予測されたメタデータ属性において、同一メタデータが何度も更新される可能性がある場合には、その属性のメタデータがメタキャッシュ42内で変更された時点ではログ採取を行わず、トランザクション44が終了した時点で、最終形態のメタデータをログとして採取する。採取した情報は、ログバッファ46に保持される。

【0058】これにより、複数回更新されたメタデータのログを更新処理の度に採取することがなくなり、メモリの効率化を図ることができるとともに、ログを書き出すときのI/Oの量も減らすことができる。

【0059】図5は、第5の発明の原理構成図である。第5の発明は、メタデータ割り当て管理情報の一部の複製を生成し、複製として生成した情報内からのみメタデータの獲得を可能とし、解放する際には、割り当て管理情報においてのみ解放された旨の情報の更新を行うことで、解放直後に別のトランザクションに獲得されるのを防止したものである。第5の発明は、以下のような要素で構成される。

【0060】割り当て管理情報保持手段51は、メタデータに対する割り当てを管理するための割り当て管理情報51aを複数の領域に分割して保持する。獲得操作管理情報生成手段52は、割り当て管理情報保持手段51内の割り当て管理情報の一部領域の複製を生成し、獲得操作管理情報51bとする。また、獲得操作管理情報51b内に未獲得のメタデータがなくなると、割り当て管理情報の別の領域の複製を獲得操作管理情報51bとする。トランザクション53は、メタデータの獲得及び解放要求を出力する。メタデータ獲得手段54は、トランザクション53によりメタデータの獲得要求が出された場合には、獲得操作管理情報51bの中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように獲得操作管理情報51bと割り当て管理情報51aとの内容を変更する。メタデータ解放手段55は、トランザクション53によりメタデータの解放要求が出された場合には、指定されたメタデータが未獲得の状態となるように、割り当て管理情報の内容を変更する。

【0061】このようなデータ処理装置によれば、獲得

操作管理情報生成手段52により、割り当て管理情報51aの一部領域の複製が生成され、獲得操作管理情報51bとされる。トランザクション53により獲得要求があると、メタデータ獲得手段54により、獲得操作管理情報51b内の未獲得のメタデータが獲得される。すると、獲得操作管理情報51bと割り当て管理情報51aとの内容が更新される。また、トランザクション53よりメタデータの解放要求があると、メタデータ解放手段55により該当するメタデータの解放処理が行われる。この際、割り当て管理情報51aの内容のみが更新される。ここで、獲得操作管理情報51b内に未獲得のメタデータがなくなると、割り当て管理情報51a内の別の領域の複製が獲得操作管理情報51bとされる。

【0062】これにより、解放された旨の情報が獲得操作管理情報51bに反映されないため、解放直後のメタデータが他のトランザクションに獲得されることがなくなる。その結果、解放処理を行ったトランザクションの終了前にシステムがダウンしても、少なくとも解放前の状態のまま保全されることが保証される。

【0063】図6は、第6の発明の原理構成図である。第6の発明は、トランザクション単位に、獲得や解放に関する情報をログとして記録することで、必要なメモリ容量の削減を図るとともに、平行動作するトランザクションのログに起因して割り当て管理情報に不正な状態が発生することを防ぐものである。第6の発明は、以下のような要素で構成される。

【0064】割り当て管理情報保持手段61は、メタデータに対する割り当てを管理するための割り当て管理情報を保持する。トランザクション62は、メタデータの獲得及び解放要求を出力する。メタデータ獲得手段63は、トランザクション62によりメタデータの獲得要求が出された場合には、獲得操作管理情報の中の未獲得のメタデータを獲得し、獲得したメタデータを獲得済みとするように割り当て管理情報61aの内容を変更する。メタデータ解放手段64は、トランザクション62によりメタデータの解放要求が出された場合には、指定されたメタデータ未獲得の状態となるように、割り当て管理情報61aの内容を変更する。ログ採取手段65は、割り当て管理情報内のメタデータ獲得手段63及びメタデータ解放手段64によって変更された部分の情報をログとして採取する。ログバッファ66は、ログ採取手段65が採取したログを保持する。

【0065】このようなデータ処理装置によれば、トランザクション62からメタデータの獲得要求が出されると、メタデータ獲得手段63によって未獲得のメタデータの1つが割り当て管理情報61a内から獲得され、そのメタデータが獲得済みの状態とされる。また、トランザクション62からメタデータの解放要求が出されると、メタデータ解放手段64によって該当するメタデー

タが未獲得の状態に変更される。そして、ログ採取手段65により、割り当て管理情報61a内のメタデータ獲得手段63及びメタデータ解放手段64によって変更された部分の情報がログとして採取され、ログバッファ66に保持される。

【0066】このように、獲得、解放のログを採取する際に、トランザクションが獲得や解放を行ったという情報のみをログとして格納することで、メモリ等の領域を効率よく利用することができるとともに、他トランザクションによる獲得や解放処理の情報がログに含まれないことによって、割り当て管理情報が不正な状態となることを防ぐことができる。

【0067】図7は、第7の発明の原理構成図である。第7の発明は、ログバッファを複数設けることにより、トランザクションの独立性をいっそう高めたものである。第7の発明の構成要素は以下の通りである。

【0068】メタボリューム71は、ファイルを管理するメタデータを記憶するための二次記憶装置である。メタキャッシュ72は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段73は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ72へと読み込む。複数のトランザクション74a~74cは、メタキャッシュ72内に読み込まれたメタデータの内容を更新する。複数のログバッファ75a~75eは、ログをトランザクション毎に保持する。各ログバッファ75a~75eのサイズは一定ではなく、大きなサイズや小さなサイズが存在する。ログ採取手段76は、メタキャッシュ72内で変更されたメタデータの内容をログとして採取し、前記トランザクション毎に分けて適したサイズのログバッファ75a~75eに格納する。例えば、最初にログを格納する場合には、トランザクションの内容によって予想される処理に適した大きさのログバッファに格納し、格納対象となるログバッファの記憶容量が不足してきたら、より大きな記憶容量のログバッファへログを移し替え、以後、より大きな記憶容量のログバッファをログの格納対象とする。

【0069】このようなデータ処理装置によれば、トランザクション74a~74cがメタデータの更新をする際には、まず、メタデータ読み込み手段73によりメタボリューム71内の必要なメタデータがメタキャッシュ72に読み込まれる。ここで、トランザクション74a~74cがメタデータの内容を更新する。すると、ログ採取手段76により、メタキャッシュ72内で変更されたメタデータがログとして採取され、適したサイズのログバッファ75a~75eに保持される。

【0070】このように、複数のログバッファに分け、トランザクション毎に1つのログバッファを使用するようにしたことで、トランザクションの独立性を保つことができる。しかも、複数のサイズのログバッファを用意

し、トランザクション毎に適したサイズのログバッファを使用することにより、メモリを効率的に利用できる。

【0071】図8は、第8の発明の原理構成図である。第8の発明は、あるトランザクションのログがログバッファに入りきらない場合に、中間ログとしてログバッファの内容を書き出すものである。第8の発明の構成は以下の通りである。

【0072】メタボリューム81は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム82は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ83は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段84は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ83へと読み込む。トランザクション85は、メタキャッシュ83内に読み込まれたメタデータの内容を更新する。ログバッファ86は、ログをトランザクション毎に保持する。ログ採取手段87は、メタキャッシュ83内で変更されたメタデータの内容をログとして採取し、ログバッファ86に格納する。ログ書き込み手段88は、トランザクション85が終了した場合にログバッファ86の内容をログボリューム82に書き込むとともに、トランザクション85によるログがログバッファ86内に格納しきれない場合には、ログバッファ86内のデータを完結したログに加工し、中間ログとしてログボリューム82内に格納する。なお、中間ログを生成する際には、中間ログに対してトランザクションを実行するのに必要とされたパラメタに関する情報を付加する。ファイルシステム復元手段89は、ファイルシステム復元要求を受け取ると、ログボリューム82に格納されたログを用いて、メタボリューム81内のメタデータの不整合を修正する。このとき、中間ログを発見すると、中間ログに含まれたパラメタを用いてトランザクションを再実行させる。

【0073】このようなデータ処理装置によれば、トランザクション85がメタデータの更新をする際には、まず、メタデータ読み込み手段84によりメタボリューム81内の必要なメタデータがメタキャッシュ83に読み込まれる。ここで、トランザクション85がメタデータの内容を更新すると、ログ採取手段87によって更新後のメタデータがログとして採取される。採取した情報は、ログバッファ86に保持される。そして、ログバッファ86に格納しきれなくなるか、トランザクション85が終了すると、ログ書き込み手段88により、ログバッファ86内の情報がログボリューム82に書き込まれる。ログバッファ86に格納しきれなくなった場合には、ログバッファ86の内容を完結したログに加工し、中間ログとしてログボリューム82に格納する。そして、ファイルシステム復元要求が出されると、ファイルシステム復元手段89により、ログボリューム82に格

納されたログを用いて、メタボリューム 81 内のメタデータの不整合が修正されるとともに、中間ログまで採取した段階で停止しているトランザクションが再実行される。

【0074】これにより、メタデータの更新を大量に行うトランザクションの実行中にシステムがダウンした場合には、途中の状態まで戻すことができるとともに、トランザクションが再実行されることで、トランザクションの実行後の状態へ遷移させることができる。

【0075】図 9 は、第 9 の発明の原理構成図である。トランザクションの受け入れを一定の条件によって制限することで、メモリ枯渇等を防止するものである。第 9 の発明の構成は以下の通りである。

【0076】メタボリューム 91 は、ファイルを管理するメタデータを記憶するための二次記憶装置である。ログボリューム 92 は、メタデータの更新結果であるログを記憶するための二次記憶装置である。メタキャッシュ 94 は、メタデータを記憶するために主記憶装置内に設けられた記憶領域である。メタデータ読み込み手段 93 は、メタデータの内容が変更される際に、対象となるメタデータをメタキャッシュ 94 へと読み込む。互いの同時実行可能な複数のトランザクション 90b~90d は、メタキャッシュ 94 内に読み込まれたメタデータの内容を更新する。トランザクション受け入れ制限手段 90a は、トランザクション 90b~90d からの開始要求を受け付けると、ログ採取に関するシステムの動作状況に基づいて、トランザクション 90b~90d の受け入れ可否を判断する。受け入れ判断基準としては、例えばログボリューム内の有効なログが占める割合を用いる。すなわち、有効範囲監視手段 99 によって有効なログとされたログがログボリューム中に占める割合が一定値以上である間は、トランザクション 90b~90d の受け入れを拒絶する。

【0077】ログ採取手段 96 は、メタキャッシュ 94 内で変更されたメタデータの内容をログとして採取する。ログバッファ 95 は、ログ採取手段 96 が採取した情報を保持する。ログ書き込み手段 97 は、ログバッファ 95 が保持する情報を、適宜ログボリューム 92 に格納する。メタデータ書き込み手段 98 は、メタキャッシュ 94 内のメタデータをメタボリューム 91 内に格納する。有効範囲監視手段 99 は、メタデータ書き込み手段 98 による書き込み動作を監視しており、変更内容が前記メタボリュームに反映されていないメタデータに対応するログボリューム 92 内のログを、有効なログとして指定する。

【0078】このようなデータ処理装置によれば、トランザクション 90b~90d から開始要求が出されると、トランザクション受け入れ制限手段 90a が受け入れの可否を判断する。例えば、有効範囲監視手段 99 により有効であると指定されたログのログボリューム 92

内に示す割合が一定以上の場合には、それ以上ログが発生しないように、トランザクションの受け入れを拒絶する。その後、メタデータ書き込み手段 98 によるメタデータの書き込みが進み、有効なログの割合が減少したら、その時点でトランザクションの開始要求を許可する。

【0079】これにより、ログボリュームの空き容量の減少に伴うハングアップなどの障害の発生を防止することができる。次に、本発明の実施の形態を具体的に説明する。

【0080】図 10 は、本発明を適用するデータ処理装置のハードウェア構成図である。データ処理システムは、CPU (Central Processing Unit) 211 を中心に構成されている。CPU 211 は、バス 217 を介して他の機器を制御するとともに、様々なデータ処理を行う。バス 217 には、メモリ 212、入力機器インタフェース 213、表示制御回路 214、HDD (Hard Disk Drive) インタフェース 215、及びネットワークインタフェース 216 が接続されている。

【0081】メモリ 212 は、CPU 211 が実行すべきプログラムや、プログラムの実行に必要な各種データを一時的に保持する。入力機器インタフェース 213 は、入力機器としてキーボード 221 とマウス 222 が接続されており、これらの入力機器からの入力内容を CPU 211 に伝える。

【0082】表示制御回路 214 は、表示装置 223 が接続されており、CPU 211 から送られてきた画像データを表示装置 223 で表示可能な画像情報に変換し、表示装置 223 の画面に表示させる。

【0083】HDD インタフェース 215 は、複数の HDD 231~233 が接続されており、CPU 211 から送られてきたデータを HDD 231~233 に格納するとともに、CPU 211 からの要求に応じて HDD 231~233 内のデータを読み取り、CPU 211 に転送する。

【0084】ネットワークインタフェース 216 は、LAN (Local Area Network) に接続されており、LAN を介してデータ通信を行う。すなわち、CPU 211 から送られたデータを LAN に接続された他のコンピュータに転送するとともに、他のコンピュータから LAN を介して送られてきたデータを CPU 211 に転送する。

【0085】HDD 231~233 には、各種ファイルや、そのファイルを管理するためのメタデータ及びログが格納されている。このような構成のシステムにおいて、CPU 211 が HDD 231~233 に格納されたオペレーティングシステム用のプログラムを実行することにより、本発明のログ採取機能が実現される。

【0086】図 11 は、ファイルシステム上で動作するログ採取機能の構成図である。図のように、ファイルを管理するためのメタボリューム 111~113 も複数設

けられている。メタボリューム111～113には、それぞれメタデータが格納されている。メタデータは、ファイルの格納場所等を管理するために必要な情報を有している。

【0087】ログボリューム120は、ログ122を格納するための二次記憶装置である。ログボリューム120には、ログ122の他にボリューム管理情報121が格納されている。

【0088】メタキャッシュ130は、メタデータを操作するためのメモリ上の領域である。メタキャッシュ130内には、操作対象となるメタデータ132とそのメタデータの割り当て管理情報131とが格納される。

【0089】ログキャッシュ140は、複数のログバッファ141～144を有している。ログバッファ141～144のサイズは均一ではなく、大きなサイズのものや小さなサイズのものがある。これらのログバッファ141～144には、メタキャッシュ130内で更新されたメタデータの複製がログとして格納される。

【0090】ログキャッシュ140とは別にログライトバッファ150が設けられている。ログライトバッファ150には、トランザクションの処理が終了した時点で、ログバッファ141～144内のログが転送される。

【0091】この例では、複数のトランザクション101～103が並列動作している。このトランザクション101～103は、ファイルシステムオペレーションを分割したものである。

【0092】実際にI/Oを行うのは2つのデーモンであり、それぞれをメタライトデーモン104、ログライトデーモン105と呼ぶ。ログライトデーモン105がログをログ専用の二次記憶装置であるログボリューム120に出力する。メタライトデーモン104は、ログボリューム120に対してログが出力されたことを確認した後、そのログに対応するメタデータをメタデータ専用の二次記憶装置であるメタボリューム111～113に出力する。

【0093】さらに、本発明のログ採取機能では、以下のような特徴を有している。第1の特徴は、各メタデータに対応するメタデータ管理情報として、メタボリュームを識別する情報が付加されていることである。これは、大規模ファイルシステムに対応するためのものである。すなわち、複数の二次記憶装置がメタボリューム111～113として定義される場合に、メタキャッシュ130上のメタデータ管理情報にそのメタデータが存在するボリュームの情報を持たせている。

【0094】図12は、メタデータ管理情報を示す図である。メタデータ管理情報には、「ボリューム番号」、「メタデータ番号」、及び「メタデータポインタ」が登録されている。「ボリューム番号」には、対応するメタデータが存在するボリューム番号が登録されている。

「メタデータ番号」には、ボリューム毎におけるメタデータ管理番号が登録されている。すなわち、システムが認識するボリュームのデバイス番号とそのボリューム内の位置から算定される数値によってメタデータが管理され、メタデータ自体がそれらの値を管理情報として保持する。「メタデータポインタ」には、メタデータの実体のある場所を指し示している。

【0095】このようなメタデータ管理情報を有するメタデータの内容が更新されると、更新後のメタデータの内容がログとしてログバッファに記録されるとともに、メタデータ管理情報の内容がログに記録される。

【0096】図13は、ログバッファの形式を示す図である。ログバッファには、「BEGINマーク」、「ボリューム番号」、「メタデータ番号」、「メタデータ実体」、及び「ENDマーク」の情報を含んでいる。

【0097】ファイルシステムを復元する際には、ログ内に記されているボリューム番号によって、そのログによって復元すべきメタデータの存在するメタボリュームが認識される。これにより、従来技術のようにファイルシステム復元時にメタデータ番号からその存在すべきボリュームを決定するよりも速度向上が望め、システムダウン後の大規模ファイルシステムにおいてもログ機構導入によるファイルシステム復元時間の短縮が有効に機能する。

【0098】第2の特徴は、メタキャッシュ130内で更新されたメタデータが、それぞれの管理構造にリンクポインタを持つことである。ログとしてログボリューム120に記録されたメタデータはメタライトリストに繋がれ、メタボリュームへ反映が完了した時点でこのメタライトリストから外される。さらに、ログとして記録されているログボリューム120内の位置情報をも、メタライトリスト内に持つ。このログボリューム内位置情報をリストを辿って検索することによって、システムダウン時に必要とされるログの範囲を特定することができる。そこで、ここから得られる有効範囲情報をログボリューム120の特定位置に設けたボリューム管理情報121内に記録する。有効範囲情報を記録し、ファイルシステム復元時にそこに含まれるログのみをリプレイすることにより、ログボリューム全体を検索する必要がなくなり、ログボリューム全体を読み込む必要のある、有効範囲情報を記録しない従来の方式よりもファイルシステムの復元時間をさらに短縮することができる。

【0099】ところで、ログ機構はシーケンシャル性を持ったディスクアクセスを行うことで、記録すべき内容をヘッドシークすることなしにディスクへ保存でき、ディスクアクセス時間の短縮を図っている。ところが、本手法を採用した場合、メタデータのメタボリューム反映時、ログボリュームへの書き出し時に、ログボリュームの特定位置に有効範囲情報を書き出すこととなる。これではシーク削減の意図が全く意味を成さない。

【0100】そのため、本実施の形態ではある程度のインターバルを空けて、有効範囲情報は書き出すように工夫した。変更がある度に、常に書き込まれるわけではないため有効範囲情報として保存されている情報には若干の誤差が含まれてしまう。ファイルシステム復元時にその誤差を吸収する必要がある。

【0101】図14は、有効範囲を説明する図である。図中において、「●」で示すのが、まだメタボリュームに反映が終了しておらず、ファイルシステム復元に必要なログを意味する。「○」で示すのは、メタボリュームに反映が完了したことによって、ファイルシステム復元時には利用しなくても良いログである。

【0102】従来のファイルシステムにおいて、ログを用いたファイルシステム復元では、ログボリューム全体を検索し、ログに記録されたシーケンス番号から、最古のログを求め、たとえそれがメタボリュームに反映済みの、利用しなくても良いログであっても利用して、ログボリューム全体のログを用いていた。

【0103】本発明では、ある時点で有効範囲情報を書き出した時の有効範囲が示されている。その後、有効範囲を書き出さずに、メタボリュームへの反映が進み、また、別のトランザクションのログがログボリュームに書き出されたことによって、実際の有効範囲と有効範囲情報とはズレを生じている。この時点でシステムがダウンし、ログを用いてファイルシステムを復元する場合には、多少のムダが生じるが、有効範囲情報が示す先頭位置から、ログを利用する。利用すべきログの末端は有効範囲情報以降の一定範囲を検索しなければならないが、その検索は有効範囲情報を書き込むインターバルに依存し、範囲が限られている。

【0104】第3の特徴は、ログに対してトランザクション終了時にシーケンス番号を与え、その番号にはデータサイズが肥大化することを考慮に入れた上で、十分大きいデータ型を適用することである。データ型の大きさは、コンピュータ自身の使用可能年数の間使い続けても枯渇しない程度のサイズとする。例えば、システムの年表記を4桁の十進数「最大9999」で表していた場合、西暦1万年まで使用されることは想定されていない。その場合、西暦9999年まで使用可能なデータ型とすれば、ログに対するシーケンス番号がオーバーフローして逆転することがないことを保証することができ、それにより、ログボリューム全体をゼロクリアして初期化する必要が生じない。具体的には、データ型を64ビット型とすれば、オーバーフローすることは現実にはありえない（4万年ほど耐えられるものと思われる）。ファイルシステム管理情報、各トランザクションのログ、有効範囲情報にこのシーケンス番号を含め、ログボリュームに記録する。このように、ログに与えるシーケンス番号のデータ型に十分大きいものを適用することにより、通常の運用時にトランザクションが動作する毎にイ

ンクリメントしても、オーバーフローは現実には起こり得ない。

【0105】さらに、スーパブロックと呼ばれるファイルシステム全体の管理情報にこのシーケンス番号を含め、正常なアンマウント処理時及びファイルシステム復元時にスーパブロックに含まれるシーケンス番号を正しく設定し、次のマウント時にそこから得られる値を用いる。これにより、シーケンス番号は必ず昇順となることが保証され、ファイルシステム復元時に新しいログを古いものと取り違えないことが保証される。

【0106】シーケンス番号の順序性を保証することによって、ファイルシステム復元時にログボリュームをゼロクリアしなくとも、常に正しいログを利用することが可能となり、ファイルシステム復元時にログボリューム全体をゼロクリアするのに比べ、大幅な時間短縮が可能となる。

【0107】以上の理由により、ログの最大のメリットであるファイルシステム復元の時間短縮がより一層有効に機能することが可能となる。第4の特徴は、トランザクションによって更新されたメタデータが、そのメタデータの属性に応じ、再度同一のトランザクションにおいて更新される可能性のあるメタデータであった場合には、更新の時点ではログバッファにはコピーせず、リスト構造（トランスリスト）によって管理することである。

【0108】そして、同一トランザクション内で複数回更新されるメタデータとして、領域割当て管理情報をリスト構造で管理し、トランザクション進行中にはその中途段階のログは採取しない。トランザクション終了時にリスト構造を辿って、トランザクションの更新の最終状態のみを一括してログとすることによって、ログの縮小が図られ、それに伴いファイルシステム復元時間の短縮が可能となる。

【0109】具体的には、メタキャッシュ130内のメタデータを管理する構造にトランスリストへのリンクポインタを持たせることによって実現している。トランザクションによって更新されたメタデータは、ただ一回しか更新されないことが分かっているメタデータである場合には、その時点でログバッファへコピーされるが、以降もトランザクション進行中に更新される可能性がある場合には、このリンクポインタを用いてリスト構造に繋がれる。メタデータの更新可能性の有無は、トランザクションの種別で判断する。例えば、データ領域の空きなどを管理するためのトランザクションでは、何度もメタデータが更新される。

【0110】そして、トランザクションが終了する時にこのトランスリストを辿り、メタデータの最終形態をログバッファへコピーする。すると、結局はトランザクションが同一メタデータを複数回更新しても、最終形態の一回だけのログ採取で済まされる。

【0111】図15は、ログ採取処理のフローチャートである。この処理は、オペレーティングシステムを実行するCPUが行う処理である。以下、CPUがオペレーティングシステムを実行することにより実現する機能を、単に「システム」ということとする。

【S1】トランザクションの開始宣言を行う。

【S2】ログ採取要求を行う。

【S3】対象メタデータの更新可能性の有無を判断する。更新可能性があればステップS5に進み、そうでなければステップS4に進む。

【S4】ログバッファへメタデータをコピーし、ステップS2に進む。

【S5】対象メタデータはトランスリストに繋がれているか否かを判断する。トランスリストに繋がれていればステップS7に進み、そうでなければステップS6に進む。

【S6】メタデータ毎にトランスリストに繋ぐ。

【S7】トランザクションの処理が終了するか否かを判断する。終了するのであればステップS8に進み、そうでなければステップS2に進む。

【S8】トランザクション終了宣言を行う。

【S9】トランスリストを辿り、繋がれている最終形態のメタデータをそれぞれログバッファへコピーする。

【0112】このようにして、トランザクションの更新の最終状態のみを一括してログとして保存することができる。第5の特徴は、メタボリュームの割当て管理情報は獲得用として通常の割当て管理情報の複製を新たに設けたことである。ここで、割り当て管理情報として、ビットマップを例に挙げて説明する。

【0113】図16は、メタボリュームの割り当て管理状況を示す図である。割り当て管理情報131には、使用されているメタデータを管理するためのビットマップ131aが用意されている。ビットマップ131aは複数のブロックに分けられている。そして各ブロックのビットマップの各ビットが「0」か「1」かによって、対応するメタデータが空いているか否かが示される。そして、ブロックに分けられたビットマップの1つの複製が作られ、獲得用ビットマップ131bとされる。

【0114】獲得時には、通常のビットマップ操作と同様に未割り当て状態の領域の検索が行われる。この時、検索対象として用いるのは獲得用ビットマップ131bである。獲得用ビットマップ131b内から未割り当て状態の領域（ビットが立っていない）が見つかった時には、その獲得要求には見つかったビット番号を返す。そして、獲得用ビットマップ131b及び、その複製元となったビットマップの対応ビットを立てる。一方、獲得用ビットマップ131b内から未割り当て状態の領域が見つからなかった時には、別のブロックのビットマップの複製を生成し、新たな獲得用ビットマップ131cとする。

【0115】図17は、ビットマップによるメタデータ獲得処理を示すフローチャートである。この処理は、システムが行う処理である。

【S11】獲得要求を発行する。

【S12】獲得用ビットマップに空きがあるか否かを判断する。空きがあればステップS20に進み、そうでなければステップS13に進む。

【S13】メモリ（メタキャッシュ130）上のビットマップに「FreeDirty」フラグが立てられていないビットマップが存在するか否かを判断する。存在すればステップS14に進み、存在しなければステップS16に進む。ここで「FreeDirty」フラグとは、一回以上の解放処理が対象ビットマップに対してなされたことを意味する。

【S14】「FreeDirty」フラグが立てられていないビットマップの中で、空きのあるものがあるか否かを判断する。そのようなビットマップがあればステップS15に進み、そうでなければステップS16に進む。

【S15】「FreeDirty」フラグが立てられておらず、空きのあるビットマップの複製を、獲得用ビットマップに作成する。その後、ステップS20に進む。

【S16】メタボリューム111～113上のビットマップに空きがあるか否かを判断する。空きのあるビットマップがあればステップS17に進み、そうでなければステップS18に進む。

【S17】メタボリューム111～113上の空きのビットマップをメモリ（メタキャッシュ130）上に読み込み、ステップS15に進む。

【S18】メモリ（メタキャッシュ130）上のビットマップに「FreeDirty」フラグが立てられたビットマップが存在するか否かを判断する。存在すればステップS19に進み、存在しなければ、獲得不可能と判断し処理を終了する。

【S19】「FreeDirty」フラグが立てられたビットマップをメタボリュームに反映し、「clean」状態にする。ここで「clean」状態とは、獲得、解放の処理が全く行われていない状態を示す。その後、ステップS13に進む。

【S20】獲得用ビットマップのビットを立てる。

【S21】複製元ビットマップの同一ビットを立てる。

【S22】複製元ビットマップに「AllocDirty」フラグを立てる。ここで、「AllocDirty」フラグとは、対象ビットマップから一回以上の獲得処理によって更新がなされた場合に、そのビットマップの管理構造体内のフラグに立てる値である。「AllocDirty」フラグや「FreeDirty」フラグが立ったビットマップはログ採取対象である。メタボリュームに反映された時にこれらのフラグは落とされ、「clean」状態となる。

【0116】このようにして、空きのビットマップを獲得できる。解放時にも、通常のビットマップ操作と同様

に、要求された領域が対応するビットの算出がまず行われるが、対応するビットを落とす操作は、対象のビットマップに対してのみ行い、仮にその対象ビットマップの複製が獲得用ビットマップとして存在する場合にも、その獲得用ビットマップに対しては行わない。

【0117】図18は、解放処理のフローチャートである。この処理は、システムが行う。

【S31】解放要求を出す。

【S32】対象ビットマップがメモリ（メタキャッシュ130）上にあるか否かを判断する。対象ビットマップがあればステップS34に進み、そうでなければステップS33に進む。

【S33】メタボリューム111～113上の対象ビットマップをメモリ（メタキャッシュ130）上に読み込む。

【S34】対象ビットマップの対応するビットを落とす。

【S35】対象ビットマップに「FreeDirty」フラグを立てる。

【0118】この一見複雑に見える作業によって、解放した領域を即座に別の用途に利用してしまうことを回避することが可能となり、システムダウン時に中途までしか終了していなかった解放トランザクションが解放したはずである領域は、解放される直前の状態のまま保全されることが保証できる。

【0119】例えば、Aというトランザクションが解放処理、Bというトランザクションが獲得処理を行う場合を考える。ここで、トランザクションAの解放処理はトランザクションBの獲得処理よりも先に行われ、かつ、トランザクションAはトランザクションBよりも後に終わる場合を例に挙げる。

【0120】図19は、トランザクションの処理の開始と終了の状況を示す図である。この図において、各トランザクションは「BEGIN」で始まり、「END」で終わることを意味し、トランザクションの「○」が解放処理を、「●」が獲得処理を意味する。

【0121】上図のように、トランザクションAの解放処理がトランザクションBの獲得処理よりも先に行われ、かつ、トランザクションBのほうがトランザクションAよりも先に終了した場合に従来のログ採取方式では問題が生ずることは既に述べた。

【0122】本発明が提案する手法を用いれば、トランザクションBが獲得する領域がトランザクションAが解放した領域となることは基本的にはなく、領域枯渇状態に近く、どうしてもこの領域を獲得せねばならない時には、一旦、該当ビットマップをメタボリュームに反映した後、獲得処理が行われる。これにより、ファイルシステム復元時に必要とされる、メタボリュームに未反映なメタデータを対象とするログを用いた復元では、管理情報上、二重獲得と見えるようなログは残り得ない。

【0123】また、トランザクションBのログにはトランザクションAの解放処理が記録されないため、トランザクションBのログよりも後に復元に用いるトランザクションAのログによって、トランザクションBが獲得した領域を変更されることもありえない。

【0124】第6の特徴は、前述のようにログ機構の導入にあたり、獲得・解放操作のログとして、その獲得・解放した対象の情報（このビットを立てた・落した）のみを記録することである。これによって、複数の獲得・解放要求に対して、要求の発生後、トランザクションの終了時点までをシリアル化することなくログ管理でき、かつログ採取量は減少し、複数トランザクション動作による変更をメタボリュームへ反映する回数をも減少させることができる。

【0125】第7の特徴は、ログキャッシュ140を複数のログバッファで管理し、この時、いくつかの異なるサイズとすることである。ログキャッシュ140を分割して複数のログバッファとして管理することによって、トランザクションの独立性を確立することが容易となるとともに、有限なメモリ空間を有効に活用することが可能となる。

【0126】まず、トランザクションがメタデータを更新するより以前に、自分がメタデータを更新する旨、システムに宣言する。この時、トランザクションの種別より予測されるログ採取情報に応じて、最適なサイズのログバッファがシステムより与えられ、トランザクション進行に伴い蓄積されるログはここへ溜められることとなる。上記第6の特徴で説明した手法に加え、トランザクション毎に異なるログバッファを用いることによって、複数のトランザクション更新の混じらない、純粋なログとして残すことが可能である。

【0127】第8の特徴は、上記第7の特徴で説明した手法に加え、トランザクションの開始時に予測したログ採取量よりも多くのログを採取しなければならなかった場合に、与えられたログバッファから、さらに大きいログバッファへ移行することである。システムの状態に応じて、トランザクションによって残されるログのサイズは変動し、その差異を複雑な処理を必要とせず、かつ、有限なメモリ空間を有効に活用して吸収することが可能である。

【0128】第9の特徴は、ログバッファに入りきらない場合には、ログボリュームへ中間ログを書き出すとともに、中間ログを書き出すことによる矛盾の発生を防止する機能を備えたことである。

【0129】ログボリュームに対するI/O発行を減少させるために、ログライトバッファと呼ぶ二次キャッシュ領域を設けている。終了したトランザクションのログは、ログバッファからこのログライトバッファへ移され、適時にログライトデーモン105によってログボリュームへ書き出される。この適時とは、負荷が低い場合

には一定の周期で良いが、トランザクションによるメタデータの更新量が多く、最大のログバッファでも収まらない場合などには強制的なI/O発行が必要とされる。このような場合において、中途の段階でのログの出力では、まだ更新されていない(かもしれない)メタデータを含めた書き出しを行うことがファイルシステムの整合性を保つためには必要である。

【0130】このように有限なメモリ空間を有効活用することを考慮した上で、ログバッファ内に収まりきらないログをトランザクション途中で出力する時に、残されるログによって復元されるファイルシステムの整合性を正当なものとするための手法を提案している。

【0131】ファイルシステムに対する負荷がそれほど高くない場合には、ログのログボリュームに対するI/Oを極力少なくするために事前に与えられた周期に応じて定期的にデーモンによって行われる。しかし、ログバッファが枯渇し、以降のトランザクション進行で溜められるログが収まらないと判断された時、部分的なログとして、この時点のログをログボリュームへ出力する。この時、まだ更新されていない情報をもログに出力する。例えばファイルを追加ライトするトランザクションの場合は、部分的なログを出力する時点でのファイルサイズや時刻情報を合わせてログに記録する。これによって、部分的であるはずのこのログを、ファイルシステム復元時には完結した1つのトランザクションのログと同等に扱うことが可能となり、ファイルシステム復元時に複雑な考慮の必要なしに、望ましい状態にファイルシステムを復元することが可能となる。

【0132】図20は、ログバッファへのログの格納状況を示す図である。この例では、2種類のサイズのログバッファ141～146が設けられている。ログバッファ141～145は通常のサイズであり、ログバッファ146が大きなサイズである。

【0133】また、ログバッファ141～146を管理するためのログバッファ管理テーブル148、149が設けられている。ログバッファ管理テーブル148は、ログバッファ141～146に対応するフラグビットを有している。そして、使用されているログバッファに対応するフラグビットの値が「1」に設定され、未使用のログバッファに対応するフラグビットの値は「0」に設定されている。同様に、大きいサイズのログバッファ146に対応するログバッファ管理テーブル149もフラグビットを有しており、そのフラグビットの値によって、ログバッファ146が使用されているか否かを管理している。

【0134】ログキャッシュ140内の各ログバッファ141～145のうち、ログバッファ143とログバッファ145とが現在利用中であることを意味するために、「●▲」などの記号を用いた。ここで、ログバッファ145には多くのログが溜まっており、既に満杯の状

態となっている。このように通常のサイズのログバッファ141～145では容量が足りなくなった際には、そのログを大きいサイズのログバッファ146に移動する。そして、トランザクションが継続している間は、更新されたメタデータの内容を、随時ログバッファに格納していく。ここで、トランザクションが終了したら、そのトランザクションに対応するログバッファの内容をログライトバッファ150へ転送する。

【0135】ログライトバッファ150には、複数のバッファ151、152が設けられており、それらのバッファ151、152にログが書き込まれる。ログライトバッファ150内のログは、ログライトデーモン105によって随時ログボリュームに書き込まれる。

【0136】図21は、ログ採取手順を示すフローチャートである。これはシステムが行う処理である。

【S41】BEGIN宣言を行う。

【S42】ログバッファの予約をする。

【S43】ログ採取要求を行う。

【S44】現在のログバッファに今回のログが収まるか否かを判断する。収まるのであればステップS51に進み、収まらないのであればステップS45に進む。

【S45】現在のログバッファより大きいログバッファが存在するか否かを判断する。存在すればステップS46に進み、そうでなければステップS47に進む。

【S46】現在のバッファから大きいバッファへ内容をコピーし、ステップS44に進む。

【S47】トランザクションに与えられたパラメタをログ採取する。

【S48】現時点のファイル状態を正しく表す管理情報をログ採取する。

【S49】現在の中途段階のログをログライトデーモン105に書き出させる。

【S50】現在のログバッファをクリアする。

【S51】ログを採取する。

【S52】ログ採取要求を終了する。

【S53】END宣言があるか否かを判断する。END宣言があればステップS54に進み、そうでなければステップS43に進む。

【S54】END宣言を行い、処理を終了する。

【0137】第10の特徴は、トランザクションが完了する前に、システムダウンが発生した場合に備え、分割して出力するログにはトランザクションに与えられたパラメタを含め、ファイルシステム復元時にそのパラメタを元に、再度トランザクションを実行することである。

【0138】さらに、この部分的なログにトランザクションに与えられたパラメタを記録し、ファイルシステム復元時にそれを用いて、通常のログだけでは途中で終わった状態までしか復元できないファイルを、トランザクションが終了した状態にまでファイルシステムに反映することが可能となる。

【0139】図22は、ファイルシステム復元処理を示すフローチャートである。これはシステムが行う処理である。

【S61】システムダウン等が発生する。

【S62】ファイルシステムの異常を検知する。

【0140】以降、ファイルシステム復元処理を行う。

【S63】ログ開始マークを検出する。

【S64】ログ開始マークに対応するログ終了マークが存在するか否かを判断する。ログ終了マークが存在すればステップS65に進み、ログ終了マークが存在しなければステップS66に進む。

【S65】開始マークから終了マークまでのログを用いて復元処理を実行する。その後、ステップS63に進む。

【S66】対象トランザクションが以前のログだけで完結するか否かを判断する。完結するのであれば復元処理を終了し、完結しないのであればステップS67に進む。

【S67】ログに記録されたパラメタを読み込む。

【S68】トランザクションとして行いたかった処理を理解する。

【S69】ファイルに対して直接処理を再実行する。その後、復元処理を終了する。

【0141】これにより、オペレーションのセマンティクスを保証したファイルシステムの復元が可能となり、従来技術では完全に元に戻す、または完全に再実行することが不可能であったトランザクションを、完全に再実行することによってファイルの整合性をも回復することが可能となる。

【0142】第11の特徴は、メタデータを更新するトランザクションが更新処理を行う前に、その旨をシステムに対し宣言することである。さらに、システムは宣言を受け入れるか否かを判断する機構を持つことである。ここでは、以下の条件の場合には新規のトランザクションの受け入れを拒否し、拒否されたトランザクションは再度認可が下りるまで待たせなければならない。

- ・メタキャッシュ130がフルに近い状態にある場合。
- ・トランザクションの多重度が、システムで規定された値を既に越えていた場合。
- ・ログボリュームに残されたログの大部分が有効なログ状態である場合。

【0143】これらの場合に、ファイルシステムが新規トランザクションの受け入れを拒否し、トランザクションの多重度を制限することによって、結果としてダーティなメタデータの数を制限することが可能となり、メタキャッシュ130の空き領域枯渇などを要因とするハングアップ状態に陥ることを回避することが可能となる。特に、分割ログとしてログ採取しなければならないトランザクションが動作中に、新規トランザクションの開始を拒否することは、システムの状態を正常に維持する上

で効果が高い。

【0144】第12の特徴は、新規トランザクションの受け入れ拒否の機構をログボリューム内の有効ログの割合に応じて適用することにより、単一の、多数のメタデータを更新するトランザクションのみならず複数のトランザクションが更新したメタデータによって空き領域の減少に伴うハングアップ状態をも回避することである。

【0145】以下に、第11の特徴と第12の特徴とを含むトランザクションの受け入れ処理について説明する。図23は、新規トランザクションの受け入れ許可判定処理を示すフローチャートである。

【S71】トランザクションを行うプロセス（以下、単にトランザクションという）が、トランザクション処理を開始する。

【S72】「BEGIN」宣言を行う。この際、オペレーティングシステム（以下、単に「システム」という）に対して問い合わせを行う。

【S73】トランザクションは、システムからの回答待ち状態となる。システムより「OK」の回答を受け取ったらステップS82に進む。

【S74】トランザクションからの問い合わせを受けたシステムが、パラメタの評価を開始する。

【S75】システムは、既に多重度が規定値より上であるか否かを判断する。多重度が規定値より上であれば「NG」としてステップS81に進み、そうでなければ「OK」としてステップS76に進む。

【S76】システムは、現在サブトランザクションが動作中であるか否かを判断する。サブトランザクションが動作中であれば「NG」としてステップS81に進み、そうでなければ「OK」としてステップS77に進む。ここでサブトランザクションとは、ログを複数に分割して格納する場合に、1つのログバッファにログを格納できるような処理単位に分割されたトランザクションである。

【S77】システムは、メタキャッシュ130に十分な空きがあるか否かを判断する。十分な空きがなければ「NG」としてステップS79に進み、十分な空きがあれば「OK」としてステップS78に進む。

【S78】システムは、ログボリュームに上書き可能領域が少ないか否かを判断する。上書き可能領域が十分になれば「NG」としてステップS79に進み、十分な上書き可能領域があれば「OK」としてトランザクションに対して「OK」の回答を返す。

【S79】システムは、ログライトデーモン105を起動する。

【S80】システムは、メタライトデーモン104を起動する。

【S81】システムは、NG条件が解消されるまでスリープ状態で待つ。NG条件が解消されたら、ステップS75に進む。

【S82】トランザクションは、システムからの「OK」の回答を受け取ったら、多重度インクリメント処理を行う。

【S83】トランザクションは、「BEGIN」処理を終了する。

【S84】トランザクションは、処理に応じて、メタデータをキャッシュに読み込み、その度に空き量をデクリメントする。分割ログ化するなら他トランザクションの開始を拒否するようにシステムに依頼する。

【S85】トランザクションは、「END宣言」を行う。

【S86】トランザクションは、多重度をデクリメントする。

【S87】トランザクションは、必要に応じてログ採取を繰り返した後、自己のトランザクション処理を終了する。

【0146】次に、本発明を適用したシステムの具体的な処理内容について説明する。本発明の全ての特徴を備えたシステムにおけるログ採取手順は以下になる。ファイルシステムオペレーションを分割したトランザクションはメタデータを更新するより前に、自分がこれからメタデータを更新する旨を宣言する(BEGIN)。BEGIN時にトランザクションに対し、独立したログバッファが割当てられる。この時、既にトランザクションの並列動作数が非常に多かったりメタキャッシュ130域の利用率が高い場合には、システムによってBEGIN宣言が拒否される。BEGIN宣言を拒否されたトランザクションはその拒否理由が解消されるまで、待ち合わせしなければならない。

【0147】更新が完了したメタデータは、BEGIN時に割り当てられたログバッファへコピーされる。それと同時にメタデータ種類に応じたりストへ繋がれる。更新すべき全てのメタデータを更新し終わったトランザクションは、そこで完了を宣言する(END)。END時には、これまで溜めたログバッファの内容をログ専用の二次キャッシュであるログライトバッファへ移動し、さらに、まだログバッファへコピーされていなかったメタデータをログライトバッファへコピーする。

【0148】また、END時にメタデータの種類に応じて繋がれていたリストから、そのトランザクションが更新した全てのメタデータを「ログ待ちリスト」へ繋ぎ替える。

【0149】以上で非同期要求のトランザクションは終了することができる。トランザクションが同期要求であった場合には、ログを書き出すまで待ち合わせなければならない。

【0150】ログの書き出しは独立したデーモン(ログライトデーモン105)が行う。このデーモンの起動契機は、同期要求トランザクションによる起動要求(wakeup)、メタキャッシュ130の空き状態監視機構による起動要求(wakeup)、及びタイマである。

【0151】起動したログライトデーモン105は複数のトランザクションのログがまとめられたログライトバッファを1つのI/Oとして発行する。そして、ログ待ちリストから「書き出しリスト」へメタデータを移動する。

【0152】メタボリュームへI/Oを発行するのは、メタライトデーモン104の役割である。メタライトデーモン104は書き出しリストに繋がれたメタデータを順に非同期ライトによってメタボリュームに反映する。メタライトデーモン104の起動契機は、ログバッファの空きが少なくなった時、ログボリュームの空きが少なくなった時、及びタイマである。

【0153】以上が、メタデータを更新するトランザクションの開始から、更新されたメタデータがメタボリュームに反映されるまでの大まかな流れである。以降に、ログの採取とそのディスク反映手法、そして、メタデータのディスク反映の処理について詳細に説明する。

(1) ログの構造

(1.1) ログボリューム

ログボリュームに格納される情報は、以下のような情報である。

【0154】ログボリュームにはスーパブロック、ボリューム管理情報が先頭にある。その次にログの有効範囲を示す構造体を記録する。構造体は以下のメンバを持つ。

- ・有効ログ先頭のログボリューム内オフセット
- ・有効ログ先頭のログシーケンス番号
- ・有効ログ末尾のログボリューム内オフセット
- ・有効ログ末尾のログシーケンス番号

ただし注意しなければならないのは、ここで先頭・末尾と言っているのは、真の値ではない可能性があることをリプレイ時に考慮しなければならない点である。なぜなら、ログはボリュームをシーケンシャルアクセスすることによってシーク時間の短縮を計っているが、このようにボリュームの一部へ毎回アクセスしたのでは、そのシーケンシャル性が損なわれてしまう。そのために、ログの書き出し毎にはこの有効範囲の書き出しは行わず、数回に一回、書き出している。

【0155】これにより、上記構造体に収められたオフセットは正確ではないためにリプレイ時に検索が必要ではある。しかしながら、全体を検索するよりも大幅に検索量が減るため、利点は保持できるものと考えている。

【0156】上記以外は、全て、メタデータの更新履歴によって構成される。

(1.2) ログブロックの基本構造

ログボリューム120内に残されたログ(メタデータの更新情報)は基本的にはトランザクション単位である。これをログブロックと呼ぶ。しかし、キャッシュ域フルなどによって、1つのトランザクションを一度にまとめることができない場合は複数の分割ログに分ける。この

分割ログをサブトランザクションログと呼ぶ。

【0157】サブトランザクションログをリプレイすることによって、ファイルシステムの整合性が保てるよう、その内容は工夫されている。しかしながら、トランザクションの途中までのサブトランザクションをリプレイしたのでは、トランザクション全体が終わっていないことから、ファイルシステムとしての整合性が保てても、ファイルの中身は異常であるという状態に陥ってしまうことが考えられる。

【0158】この状態を回避するために、そのサブトランザクションに別れたトランザクションがどのようなオペレーションであったかをログに採取する（オペレーションログ）。ログリプレイ時には、サブトランザクションをリプレイした後、オペレーションログをリプレイヤが再実行することによって、トランザクション全体が終わった状態とすることが可能である。

【0159】サブトランザクション化する可能性のあるトランザクションは全体終了時に「終わったこと」をログに記録し（最終ENDマーク）、リプレイヤはその有無を調べてオペレーションログの実行を判断する。

（1. 3）ログの採取形式

メタデータの更新情報は該当するメタデータの管理構造単位で採取する。また、メタボリュームの管理構造であるビットマップはその更新部分だけのログ採取とする、スーパーブロックについては特殊であり、データ空き容量についてののみログ採取を行う。

（1. 3. 1）inode、Vデータ、空き管理情報
ファイルの管理情報であるiノード、Vデータと呼ぶディレクトリやシンボリックリンクデータや空き管理情報のメタデータ本体は、それらのI/O単位（ブロック単位）でのログ採取を行う。

【0160】これは、ファイルシステム整合性チェック処理であるfsckによるログリプレイ時の処理を簡略化することを意識している。変更された部分だけをログ採取した場合、ログリプレイ時には、該当ブロックの算定→読み込み→変更部分の更新→再度書き込み、とステップが増えてしまう。しかし、ブロック全体のログ採取によって、リプレイ時にはメタボリュームにログ情報を上書きするだけでよい。

【0161】iノード本体やディレクトリブロックなどのVデータは、更新中はファイルのロックで保護されている。しかしながら、空き管理情報についてはファイルのロックでは保護されない。そのため、トランザクション途中で他トランザクションによって更新され、そのトランザクションが追い越して先に終わってしまうと、ログには古い情報が後に残されてしまい、リプレイするとファイルシステムに不整合が生じてしまう。

【0162】そのため、空き管理部については、ここでは更新するトランザクションが並行動作しないことを保証することによって実質的な排他制御を行い、他のメタ

データと同じ採取形式とした。

（1. 3. 2）ビットマップ

メタデータの割り当て状況を管理するビットマップも空き管理部と同様にファイルのロックで保護されていない。そのため、ビットマップ全体のログ採取を行うと、そのログには複数のトランザクションによる更新が含まれてしまう可能性がある。特に、トランザクションの追い越しが発生すると、新しいはずのログによって、ビットマップの情報が古く書き戻されてしまうことが考えられる。

【0163】そこで、ビットマップのログ採取は更新部分だけとする。更新部分とは、あるビットが0になった、あるいは1になったと記録するだけである。それにより、トランザクションが並行動作してもそれぞれのトランザクションが更新した内容のみがログに残されるため、メタデータの後退は起こり得ない。

【0164】注意すべきは、ある領域を解放（「1→0」）した後、他トランザクションがそこを獲得（「0→1」）した場合である。トランザクションの追い越しが発生すると、ログリプレイによって獲得した領域を解放してしまう。

【0165】これについては、アロケート用ビットマップを1つ定め、複製を用いて操作を行うことによって回避する。

（1. 4）ログの記録構造

（1. 4. 1）一般ログ

ログ有効状態であっても、ある程度の複数スレッドが同時に進行することを許す。これは性能要件より必須である。しかし、ログボリューム内のログは前述の通り、トランザクション毎に保存する。1つのトランザクション結果をログに記録する際、ログは以下のパーツで構成する。

1) BEGINマーク

2) ヘッダ

3) 更新内容

（2）+ 3）の繰り返し

4) ENDマーク

これを以下、ログブロックと呼ぶ。ログブロックはログボリュームの物理ブロック境界から始まるが、その終わりは物理ブロック境界であるとは限らない。

1) BEGINマーク

トランザクションの開始時に作成される情報であり、以下の内容を含む。

・マジックワード

・トランザクションタイプ

・ログシーケンス番号

・ログブロックサイズ

マジックワード；1つのトランザクションが始まったことを示すマジックワードを埋め込む。

【0166】トランザクションタイプ：BEGINマー

クにトランザクションのタイプ(種類)を埋め込むことによって、その後の更新情報に含まれる内容の概要を事前に知ることが可能。

【0167】ログシーケンス番号：ログに記録されたトランザクション毎にインクリメントされる数値。このカウンタが最小のログブロックがそのログボリューム内で最も古いトランザクションであることを示す。カウンタは64ビット型とし、オーバーフローすることは現実にはありえない(4万年ほど耐えられるものと思われる)。ログリプレイ時に、ログボリュームをゼロクリアすることにより、再度0から開始することも考えられるが、リプレイ時間の短縮を考慮し、利用したログの最後のシーケンス番号より昇順に採番することによりゼロクリアを回避する。

【0168】ログブロックサイズ：ログブロックのENDマークまでのサイズを記録する。BEGINマークの先頭からENDマークの先頭までのサイズである。リプレイ時にはBEGINマークの先頭からこのサイズだけ移動し、ENDマークの情報を参照して、ログの有効性を判定する。

2) ヘッド

更新されたメタデータについて、その保管位置を特定するための情報であり、以下のメンバによって構成される。

- ・メタデータタイプ
- ・メタボリューム番号
- ・ボリュームローカルメタデータ番号

メタデータタイプ：更新情報の後方にあるメタデータ更新内容が何のメタデータであるのかを特定するために用いられ、リプレイはここから更新内容のサイズを判断する。

【0169】メタボリューム番号：メタデータ管理で用いているメタボリューム毎の番号を記録する。リプレイ時には、この番号から書き戻すメタボリュームを決定する。

ボリュームローカルメタデータ番号：メタデータ管理で用いている、メタボリューム毎、メタデータ毎に0から始まる値を記録する。リプレイ時には、この番号からメタボリューム内のブロック位置に変換し、更新内容を書き戻す。これはブロック位置変換を削減することによるログ採取の高速化が狙いである。対象がビットマップである場合には、変更されたビットが該当するメタデータ本体のメタデータ番号を記載する(ビットマップ番号ではない)。

3) 更新内容

メタデータ本体の場合には、更新内容が含まれるブロック(それぞれのメタデータの管理単位)である。例えば、ディレクトリブロックが更新された場合には1024バイトのディレクトリブロック全体である。

【0170】ビットマップの場合には、全体ではなく、

ここには「0→1」または「1→0」という情報だけ記録する。リプレイ時にはヘッドから該当するビットマップを判定し、それを読み込み、この内容から該当ビットを変更して書き戻す。

4) ENDマーク

BEGINマークに対応するマジックワード、ログシーケンス番号、ログブロックサイズを記録する。

【0171】ENDマークがマジックワードとログ番号だけでは、メタデータの内容によってはそれが偽りのENDマークとして見えてしまい、リプレイの時に処理を誤る可能性がある。これを回避するために、ENDマークは固有形式(メタデータを誤認することのない形式)としなければならない。

【0172】BEGINマークを固有形式としないのは、BEGINマークだけ書き出された時点で、システムダウンした場合を考慮し、いずれにせよENDマークの識別ができなければならないからである。

【0173】固有形式とするために、固有の数値を64バイト分続ける。これにより、全てのメタデータがENDマークに化けることはなくなる。これに続けて、マジックワード、カウンタ(ログ番号)、ログブロックサイズを記録する。以降、ブロック境界まで、上記数値を埋める。BEGINマークに対応したENDマークを見つけてできないログ情報はリプレイしてはならない(ログ書き出し途中でシステムが異常終了したことを意味する)。

(1. 4. 2) 巨大トランザクションのログ

トランザクションが多く、メタデータを更新する場合には、ログバッファサイズ、ログボリュームサイズが有限であることから、複数のサブトランザクションログに分割する。サブトランザクションログはそれだけのリプレイによって、ファイルシステムの整合性は保たれる。しかし、トランザクション全体のサブトランザクションログをリプレイしなければ、ファイルの整合性が保てない。そのため、サブトランザクション途中でのシステムダウンを考慮し、オペレーションログを内部に含む。

1) BEGINマーク

2) オペレーションログ

3) ヘッド

4) 更新内容

(3) + 4) の繰り返し)

5) ENDマーク

(1) ~ 5) の繰り返し)

(5') 最終ENDマーク)

1) BEGINマーク

一般ログのBEGINマークと同じ。ただし、サブトランザクションログとなりうるトランザクションは、書き込みや削除など、データ領域を触るものや、領域管理に関わるものに限定される。

【0174】これらのトランザクションがBEGINマ

ーク内で設定されていたら、そのログはサブトランザクション化している可能性があり、必ずオペレーションログが続いている（たとえ単一のサブトランザクションログで構成されていても）。

2) オペレーションログ

サブトランザクション化したトランザクション（関数）に与えられた引数をオペレーションログとして保存する。巨大トランザクションを対象としたBEGINマークの直後には必ずオペレーションログを配置する。オペレーションログは、最初にトランザクションへ渡された情報（このファイルをこれだけのサイズにランケットするとか、このファイルをこれだけアペンドライトするなど）である。

【0175】リプレイ時には、リプレイヤがファイルシステムを直接操作して、ここで残されたオペレーションログを実行しなければならない。オペレーションログは以下のメンバで構成される。

- ・パラメタ1
- ・パラメタ2
- ...

リプレイヤはBEGINマーク内のトランザクションタイプを見ることによって、オペレーションログ部分に並ぶパラメタ群のサイズ及び内容を知ることができる。

3) ヘッダ

一般ログのヘッダと同じ。

4) 更新内容

一般ログの更新内容と同じ。

5) 5') ENDマーク、最終ENDマーク

一般ログのENDマークと同じ意味合いを持ち、BEGINマークに対応するENDマークあるいは最終ENDマークが見つからない場合には、BEGINマーク以降の更新内容をリプレイしてはならない。

【0176】巨大トランザクションがサブトランザクションに分割されず、単一のログブロックのみの場合には、ENDマーク部分には最終ENDマークが書き出される。複数のサブトランザクションログに分かれている場合には、最後のログブロックだけ、ENDマークが最終ENDマークに化ける（最後以外のログブロックには通常のENDマークが書かれている）。

【0177】ENDマークは一般ログのENDマークと同一であり固有数値64バイト、マジックワード、カウンタ（ログ番号）、ログブロックサイズが記録されており、ブロック境界まで固有数値が埋まる。

【0178】最終ENDマークとENDマークの差は、マジックワードのみである。最終ENDマークがトランザクション全体の終了を表すことから、とても巨大なトランザクションが多くサブトランザクションログに分割されている場合には、全ての処理が完了する時にこの最終ENDマークを出力する。すなわち、巨大トランザクションとして定義されるトランザクションについて、

通常のENDマークで終わるログブロックがいくつか見つかったのに、最終ENDマークで終わるログブロックが存在しない場合には、それは巨大トランザクションの途中でシステムダウンが発生したことを意味し、ログリプレイヤがオペレーションログのリプレイを行わなければならないことを意味する。

(2) ログの採取

(2.1) ログバッファの管理

ログをトランザクション単位にログボリューム120へ出力するためには、メタデータの更新情報を一度、メモリ内に溜める必要がある。これをログバッファと呼ぶ。ログバッファはマウント時にまとめて用意し、ファイルシステム利用中にメモリの追加獲得は行わない。

【0179】トランザクション毎のログ採取とし、また、トランザクション動作中に他のトランザクションが並行動作することを狙い、ファイルシステム毎にログバッファを複数持つ。その数は並行動作を許すトランザクション数によって決められる。

【0180】並行動作数を限定することはログ機能追加による性能劣化要因の1つとなるが、

- ・ログリプレイ時の処理を単純化
 - ・巨大なログバッファを分割して使うことによる複雑さの回避
- などのメリットが考えられるため、この方式とする。

【0181】各トランザクションは開始時に割り振られたログバッファにログを作成し、トランザクション終了時にまとめてログライトバッファへコピーする。ログライトバッファの内容を実際にI/O出力するのはログライトデーモン105と呼ぶデーモンの働きによる。

【0182】巨大トランザクションには、一般トランザクションが用いるログバッファよりも大きいログバッファ（以降、それぞれを一般ログバッファ、巨大ログバッファと呼ぶ）を1つ用意する。巨大トランザクションも当初は一般ログバッファの1つを用いる。巨大トランザクションとログバッファの関係には次の二段階がある。

1) あまり多くのメタデータを更新せず、一般ログバッファで十分足りると判断できた時点で、次の巨大トランザクションのBEGINを受け付ける。

2) 更新するメタデータ数がある程度多く、一般ログバッファでは足りない判断できた時点で、巨大ログバッファにこれまでの内容をコピー、そこで処理を継続する。

3) 更新するメタデータ数が大変多く、巨大ログバッファでは足りない場合には、サブトランザクションとして分割する。

【0183】2) におけるバッファ間のコピーが負担となりそうであるが、巨大トランザクションがそれほど多くないと考えると、あまり頻繁に発生するものではないので、この方法とする。

【0184】一般ログバッファは状態（利用中・未利

用)について、ビットマップで管理される。各ログバッファを管理する構造をログバッファ数の配列で確保し、それぞれが

- ・ログバッファのアドレス
 - ・これまでに採取したログサイズ
- を持っている。

【0185】また、ログバッファの内容は専用のログライトデーモン105によってログボリューム120へ反映するが、そのログライトデーモン105が複数のログバッファの内容を一括してI/O発行するために、トランザクション終了時にそれぞれのログバッファの内容をログライトバッファ150と呼ぶ専用の領域にコピーする。

【0186】ログライトバッファを管理する構造として、

- ・追加モードにあるログライトバッファはどちらか
 - ・それぞれのログライトバッファに溜められたログサイズ
- がある。

【0187】ログライトバッファの内容をログボリューム120へ書き出している間は、その後のトランザクションが終了するためにログライトバッファにログバッファをコピーしようとしても、I/O中であるためにガードしなければならない。これは性能劣化に繋がるため避けなければならない。そのため、ログライトバッファは2本用意する。

【0188】一般トランザクションはトランザクション開始時に、空のログバッファをビットマップより見つけ(未利用ログバッファを0で示し、ここで1とする)、そのビット番号がトランザクション番号となる。

【0189】巨大トランザクションが一般ログバッファにて処理進行中、一般ログバッファでは入りきらないと判断されるとこれまでの内容をこの巨大ログバッファへコピーし、そこで処理を継続する。この時、これまで用いていた一般ログバッファは空き状態として、次のトランザクションによる利用を可能にする。

(2.2) 巨大トランザクションの管理

空き領域管理ツリーや獲得済領域管理ツリーを操作するトランザクションを巨大トランザクションと定義する。巨大トランザクションは場合によっては木構造を大きく変更しなければならないかもしれない。この時、サブトランザクション化する必要が生ずる。

【0190】サブトランザクション化した巨大トランザクションが並行動作した場合、どれが多くメタデータを更新すると、メタキャッシュ130がいずれ全てダーティとなり、新たにメタデータを読み込んで更新しようにも追い出すこともできずにデッドロックに陥ることが考えられる。

【0191】そのため、巨大トランザクションについては並行動作ができないよう、ここではシリアルライズし

た。しかし、上記の通り、多くのメタデータを更新する可能性があると考えられるトランザクションは多い。これらを全てトランザクション終了までシリアルライズすると、その性能インパクトは大きい。

【0192】巨大トランザクションも、場合によってはそれほど多くのメタデータを更新しないこともありえる。一般トランザクションと同等の数しかメタデータを更新しないのであれば、扱いを一般トランザクションと同様にすることによって、その性能インパクトを小さくすることが可能である。

【0193】そこで、巨大トランザクションが一般トランザクションと変わらない程度しかメタデータを更新しないと分かった時点で、この巨大トランザクションの扱いは一般トランザクションと同じにする(デグレード)。

【0194】巨大トランザクションはその開始時には一般ログバッファの1つを一般トランザクション開始時と同様に与えられる。しかし、1つの巨大トランザクションが動作中は次の巨大トランザクションにログバッファを与えない点が一般トランザクションの場合と異なる。

【0195】巨大トランザクションがある程度以上のメタデータを更新することが分かった時点で、これまでの一般ログバッファの内容を巨大ログバッファへコピーし、そこで処理を継続する。

【0196】しかし、それほど多くのメタデータを更新しないと判断されれば、そのまま一般ログバッファで処理が継続され、また、次の巨大トランザクションに対し、処理の開始(ログバッファの使用)を許可する。

【0197】こうして、巨大トランザクションを途中から一般トランザクションとして扱うこと(デグレード)を実現する。

(2.2.1) デグレード契機・サブトランザクション化契機

巨大トランザクションは複数のサブトランザクションに分割される場合もあれば、一般トランザクションへデグレードする場合もある。それぞれの判定基準は以下の通りである。

◎デグレード判定基準

・一般ログバッファの残りサイズで、将来全てのメタデータ更新が完了できると判断できた場合。

◎巨大トランザクション用ログバッファへの移行契機

・一般ログバッファの残りサイズが次のステップの最大変更量よりも少ない場合。

◎サブトランザクション判定基準

・巨大ログバッファの残りサイズが次ステップの最大変更量よりも少ない場合。

(2.3) 一般トランザクションのログ採取

一般トランザクションは、以下の手順に従いログを採取する。

1) LOG_BEGIN

a) 利用するログバッファを確定する。

【0198】b) ログバッファの先頭にBEGINマークを作成する。

2) 各メタデータの更新

c) 更新されたメタデータがリリースされる（ロックが解放される）直前に、更新されたメタデータについて、ヘッダ（更新情報）をログバッファに作成し、更新内容をログバッファにコピーする。

【0199】d) 更新されたメタデータをリストに繋ぐ。

3) LOG_END

e) ENDマークをログバッファに作成する。

【0200】f) ログバッファをログライトバッファへコピーする。

g) ログライトデーモン105がログボリューム120へ反映する。

h) このトランザクションで立てられた追い出し不可フラグを落とす。

(2.3.1) LOG_BEGIN

トランザクション開始を意味する。同一関数内にLOG_ENDの宣言が必要。並行動作可能数だけ既にトランザクションが動作していたら、それらのどれか1つが終了するまで寝て待つ。

【0201】a) 現在実行中のトランザクション数がログバッファの数と等しい場合、それは既に許された並行動作数だけトランザクションが実行中であることを意味する。そのため、他トランザクションが終了するまで寝る。そのような状態でなければ、ログバッファの利用状況を管理するビットマップより未利用状態のログバッファを1つ選び出し、そのビットを立てる。さらに、並行動作数カウンタをインクリメントする。

【0202】b) 前述のBEGINマークをログバッファの先頭に作成する。

(2.3.2) 各メタデータの更新

メタデータを参照した場合にはログ採取の必要はない。更新した場合のみログを採らなければならない。

【0203】c) メタキャッシュ130に読み込んだメタデータについて処理が終了し、メタキャッシュ130より解放するための関数（リリース関数）を呼び出す際、「更新した」ことを明示された場合、ログを採取する。

【0204】i ノードについてはログ対象トランザクション内でのロック解放時がログ採取契機である。具体的には、上記で使用权を得たログバッファにヘッダを作成し、メタデータの更新内容をコピーする。このとき、メタデータの種類によってその手法が若干異なる。

【0205】c-1) ビットマップの場合：ヘッダには獲得（解放）したメタデータ番号を入れ、更新内容は「0→1」「1→0」とビット操作だけとする。

c-2) メタデータ本体の場合：更新内容にはメタデー

タをそのままコピーする。

【0206】c-3) スーパーブロックの場合：巨大トランザクションがデグレードした場合のみ、一般ログ内でのスーパーブロックのログ採取がありうる。その時刻（シークエンシャル番号）とデータ空きサイズを記録する。

【0207】コピーするとともに、ログバッファのターミナルで管理されるログサイズに、ここで作成したログのサイズを加える。

d) メタキャッシュ130の大きさは有限であることから、トランザクションが進行するためには、必要のないメタデータをメタキャッシュ130から追い出し、その場所に必要メタデータを読み込むという処理を行う追い出し機構がある。更新されたメタデータはログを書き出すまでは追い出されてはならない。そのために、このメタデータをメタライトリストと呼ぶ、メタボリュームへの反映を司るメタライトデーモン104が参照するリストへ繋ぐ。

(2.3.3) LOG_END

トランザクションの終了を示す。ここで、ENDマークの作成を行う。ログボリューム120への反映はログライトデーモン105による作業である。

【0208】e) LOG_BEGINにて作成したBEGINマークに対応するENDマークをログバッファの末尾に作成する。

f) 利用したログバッファの内容を追加モードにあるログライトバッファにコピーする。この時にログ番号を決定し、BEGINマーク、ENDマークに埋め込む。ログバッファのターミナルに記録していたログのサイズをログライトバッファのサイズに加える。ログバッファの「ターミナル」とは、そのログバッファ自身の構造に関する情報を格納する場所を意味している。同期書き込み要求の場合は、ログライトデーモン105（詳細後述）が正常にログボリューム120に反映したことを待つ必要があるが、同期書き込み要求でない場合には、ログバッファを解放（ビットを落とす）して終了する。

【0209】g) 後述するログライトデーモン105が、ログライトバッファの内容をログボリュームへ反映する。

h) リリース時に立てた追い出し不可フラグを全て落とす。ただし、フラグを落とすのはログライトデーモンによってなされるため、このトランザクションはそのまま終了する。

(2.4) 巨大トランザクションのログ採取

巨大トランザクションも、一般トランザクションとほぼ同様の処理だが、最大の違いは、トランザクションの状況によってログバッファをサイズの大きい巨大ログバッファへ移行したり、サブトランザクションとして分割したログ採取を行わなければならない場合があることである。

1) LOG_BEGIN

a) 利用するログバッファを確定する。

【0210】b) ログバッファの先頭にBEGINマークを作成する。

c) BEGINマーク作成と同時に、オペレーションログをログバッファに作成する。

2) 各メタデータの更新

メタデータがリリースされた時に「更新したこと」が明示されたら、

d) 更新メタデータが空き管理情報の場合、更新メタデータがリリースされる度にBTFリストあるいはBTAリストとよぶ空き管理メタデータのために用意するリストに繋ぐ。この時、既にリストに繋がれていればリスト構造には手を加えない。

【0211】e) 更新メタデータが空き管理情報以外の場合、更新されたメタデータがリリースされる度にヘッダをログバッファに作成し、更新内容をログバッファ域にコピーする。

【0212】f) 更新メタデータをリストに繋ぐ。まだ一般ログバッファで処理している場合、

g) 一般ログバッファで足りるか判定する。

【0213】g-a) 足りないなら巨大ログバッファへこれまでの内容をコピー。

g-b) 足りると確定でき、かつ、デグレード条件を満たせば、次の巨大トランザクションを受け付ける。

【0214】g-c) まだ判断ができなければそのまま継続する。既に巨大ログバッファで処理している場合、

h) サブトランザクション化するか判定する。

【0215】h-a) するなら、BTFリスト及びBTAリストをたどりながらそれらをログバッファにコピー、ENDマークを作成し、ログライトバッファへコピーする。ログライトデーモン105を起動し、ログを出力する。

【0216】h-b) まだしないなら、そのまま継続する。

3) LOG_END

i) 最終ENDマークをログバッファに作成する。

【0217】j) ログバッファをログライトバッファへコピーする。

k) ログライトデーモン105がログボリューム120へ反映する。

l) このトランザクションで立てられた追い出し不可フラグを落とす。

(2.4.1) LOG_BEGIN

巨大トランザクションも最初は一般ログバッファを用いる。

【0218】a) 一般トランザクションの場合は、現在の並行動作トランザクション数とログバッファの数との関係によってログバッファを獲得できるかどうか定まった。巨大トランザクションの場合は、現在他の巨大トランザクションが動作中か否かが問題であり、一般トラ

ンザクションの並行動作数とは関係しない。巨大トランザクションが動作中か否かのフラグを調査し、非動作中であれば、そのフラグを立て、ログバッファの利用状況を管理するビットマップより未利用状態のログバッファを1つ選び出し、そのビットを立てる。巨大トランザクションが現在動作中であれば、終了まで寝て待つ。

【0219】b) BEGINマークを作成する。ここで、トランザクションタイプに巨大トランザクションとなりうるトランザクションが指定されていた場合には、必ず後ろにはオペレーションログが付随する。対応するENDマークがないログはリプレイしてはならない。また、サブトランザクション化しているのに、最終ENDマークがないなら、オペレーションログのリプレイが必要である。

【0220】c) 巨大トランザクションをサブトランザクションログとして分割する場合にはオペレーションログを採取する必要がある。ここで、オペレーションログとは、各ログ採取対象関数に与えられたパラメータが、将来リプレイすることが可能な形に変更されたものである。具体的には、メモリアドレスで与えられるパラメータは、メタボリューム内のオフセットに変更して記録する。

(2.4.2) 各メタデータの更新

d) 更新メタデータが空き管理情報である場合、一回のトランザクションで同一の領域が何度も変更される場合が考えられる。その都度、ログを採取しているとログバッファやログボリュームがフルになる可能性が高まる。これを回避するために、更新情報をリストによって管理し、複数回更新された空き管理メタデータを1つにまとめてログに記録する。具体的には、メタデータの状態毎にリスト管理する専用のBTFリスト及びBTAリストに繋がれる。このリストに繋がれていれば、そのデータはダーティであることを意味する。初めて更新するデータであれば、ターミナルから繋がるリストに追加する。既にリストに繋がれているのであれば、2回目以降の更新であることを意味し、ポインタについてはそのままが良い。リストにメタデータを追加する場合には、ログサイズを更新する。

【0221】e) 更新メタデータが空き管理情報以外であれば、同一メタデータに対して何度もホールド/リリースが発行されるとは考えられないので、リリースの度(ロック解放の度)にログバッファへコピーする(一般トランザクションの場合と同じ)。ログサイズを更新する。

【0222】f) 既に何度か述べたが、更新されたメタデータがログよりも先にメタボリューム111~113へ反映されてはならない。この追い出し不可状態もリスト構造によって管理される。

【0223】g) この巨大トランザクションが更新するメタデータが、以降、どれだけの数のメタデータを更新

するかを算出することによって、処理が分かれる。

g-a) 現在利用している一般ログバッファの残りでは次ステップの実行によるメタデータの更新の全てをまかないきれない場合があると判断した場合には、巨大ログバッファへの移管を行う。

【0224】g-a-1) これまでに溜まったログバッファの内容を巨大ログバッファへコピーする。この時、BTFリストはそのまま繋いだままとし、BTAリストは巨大トランザクション用のターミナルへ移動する。

【0225】g-a-2) 巨大ログバッファを利用して、その巨大トランザクションは処理を継続する。

g-b) 現在利用している一般ログバッファの残りだけで、以降のメタデータ更新を全て記録できると判断できるならば、この巨大トランザクションは一般トランザクションにデグレードする。BTFリストにメタデータが繋がれている場合はデグレード条件を満たさないため、ここでの処理はありえない。BTFリストはそのまま利用を続ける。

【0226】g-b-1) 巨大トランザクションが動作中か否かを示すフラグを落とす。

g-b-2) 一般トランザクションの並行動作数カウンタをインクリメント。

g-b-3) 次の巨大トランザクションが寝ているならば起こして、実行を受け付ける。

【0227】g-b-4) このまま一般ログバッファを利用して処理を継続する。

g-c) まだ判断できないのであれば、このまま一般ログバッファを利用して処理を継続する。

【0228】h) サブトランザクション判定基準(前述)に基づき、このトランザクションをサブトランザクション化するかどうか判定する。

h-a) サブトランザクションに分割する場合は、

h-a-1) 更新された空き管理情報を、リストを辿りながらログバッファへコピーする。

【0229】h-a-2) ENDマークを作成する。

h-a-3) ログライトバッファへコピーする。

h-a-4) ログライトサイズを更新する。

【0230】h-a-5) ログライトデーモン105を強制起動する。

h-a-6) ここで変更した全てのメタデータの追い出し不可フラグを落としてメタライトリストに繋ぐ。これにより、メタライトデーモン104は任意の時にこれらメタデータを書き出すことが可能となる。

【0231】h-a-7) 巨大ログバッファの先頭にBEGINマークを作成する。

h-a-8) オペレーションログを作成する。

h-b) サブトランザクションに分割する必要がない間は、巨大ログバッファ内でそのまま継続する。

(2.4.3) LOG__END

トランザクションの終了を示す。ここで、最終ENDマ

ークの作成を行う。実際のログボリューム120への反映はログライトデーモン105の仕事である。

【0232】i) 最終ENDマークを作成する。最終ENDマークは通常のENDマークとマジックワードが異なるだけである。

j) 利用したログバッファの内容を追加モードにあるログライトバッファへコピーする。この時にログ番号を決定し、ログサイズをログライトサイズに加える。同期書き込み要求の場合は、ログライトデーモン105が正常にログボリューム120に反映したことを待つ必要があるが、その他の場合には待たないで次の処理へ進む(すなわち非同期書き出しである)。巨大トランザクション動作中フラグを落とし、寝て待つ次の巨大トランザクションを起こす。

【0233】k) ログライトデーモン105が、ログライトバッファの内容をログボリュームへ反映する。

1) 更新したメタデータをログバッファへコピーした時に立てた追い出し不可フラグを全て落とす。ただし、フラグを落とすのはログライトデーモン105によってなされるため、このトランザクションはそのまま終了する。

(2.5) ログライトデーモン

並行して動作し、順次終了するトランザクションによって更新されたメタデータのログは専用の書き出しデーモン(ログライトデーモン105)によってログボリュームへ反映する。このデーモンはマウント時に起動され、アンマウント時に停止する。すなわち、ファイルシステム毎にスレッドを生成する。

【0234】各トランザクションが終了すると、ログバッファ内にENDマークが作成され、このログバッファの内容はログライトバッファ150にコピーされる。このログライトバッファ150は複数のログバッファの内容を一括してログボリューム120に反映するためのものであり、そこにはメモリコピーの負担があるが、何度もI/Oを発行するよりは良いと考える。

【0235】ログライトバッファ150へコピーされると、そのログバッファは利用中バッファリストから空きバッファリストへと繋ぎ直され、かつ、トランザクションが同期書き込み要求でなければ、動作中トランザクション数をデクリメントする。これにより、次トランザクションの実行が進むようになる。

【0236】ログライトデーモン105はログライトバッファの内容を、定期的、あるいは同期書き込み指定のトランザクションがある場合などにログボリュームに反映する。反映している間(I/O中)は2本あるログライトバッファのうち、もう片方のログライトバッファに内容をコピーしてそのトランザクションは処理を終了する。

(2.5.1) 処理手順

ログライトデーモン105単体の処理手順はそれほど複

雑ではない。

【0237】a) 現在のログライトバッファをライトモードにし、もう片方を追加モードにする。

b) ログライトバッファの内容をログボリューム120へ同期書き出しする。

【0238】o) 場合に応じて有効範囲情報を書き出す。

d) ログ待ちリストに繋がるメタデータを、メタライトリストへ繋ぎかえる。

e) 規定時間の間、スリープする。

【0239】f) 規定時間が経過、または他の要因で起こされたらa)へ。

以下、詳細説明である。

a) I/O中はその対象域に対して追加更新は許されない。そのため、現在のログライトバッファが書き出しが終わり、次の書き出しが始まるまでは、もう片方のログライトバッファへ終了したログバッファをコピーするよう設定する。I/O中のログライトバッファをライトモード、ログバッファの内容をコピーする方を追加モードと呼ぶ。切り替えはこのデーモンによって行われ、各トランザクションは常に追加モードにあるログライトバッファに自ログバッファの内容をコピーする。

【0240】b) ログライトバッファに新しい内容が含まれていないのであれば、I/Oを発行する必要はない。そこで、まずログライトサイズを調べ、ゼロであればI/Oを発行せず、タイマを指定して再び寝る。書き出すべきログが存在するのであれば、以下の手順に従う。

【0241】b-a) ログボリュームの空きサイズ判定。ログボリュームの先頭オフセット(A)、メタライトリスト先頭のメタデータを管理する構造の上書き可能オフセット(B)を調べる。それとログライトオフセット(C)、及びログボリュームの最終オフセット(D)から以下の手順となる。

・ $A < B \leq C < D$

①ログライトサイズが(D-C)以下ならば、Cから書き出す。

②ログライトサイズが(D-C)より大きく、かつ、(B-A)以下ならば、Aから書き出す。

・ $A \leq C < B \leq D$

③ログライトサイズが(B-C)より小さければ、Cから書き出す。

【0242】これらの条件に合致せず、ログの出力ができない場合には、メタライトデーモン104を起動して、自分はスリープする。メタライトデーモン104の動作により起動されたら、再度、空きサイズ判定から行う。

【0243】b-b) ログボリューム120へ同期書き出しする。

b-c) エラー判定。同期書き出し後、エラー判定を行

う。エラーが発見された場合の処置については省略する。

【0244】b-d) 領域判定。

書き出しが終わった後、再び、b-a)と同様な空き領域の判定を行う。ここで、残りが少ないと判定された時、その残りの大きさに応じてメタライトデーモン104を「平常起動」または「緊急起動」する。

【0245】c) ファイルシステム復元に必要なログは、メタライトリスト先頭の上書き可能オフセットからたった今書き出した位置までである。そこで、それを有効範囲情報としてディスクへ書き出す。ここで、書き出しはログのシーケンシャル性を考慮し、ある程度の間隔をおいて行う。ここでは、回数に応じて、数回に一回、書き出すこととした。

【0246】d) b)にて書き出したログに含まれるメタデータは全て「ログ未反映状態」、すなわち追い出し不可状態としてリスト(ログ未反映リスト)に繋がれているはずである。そこで、ログ未反映リストを辿りながら、そこに繋がるメタデータをメタライトリストに追加することにより、メタライトデーモン104が任意の時にこれらのメタデータをメタボリュームに反映できるようになる。

【0247】e) ログもシステム全体から見れば非同期書き出しとし、トランザクションが同期書き込み要求でなければ、ログを書き出す前にそのトランザクションは終了することができる。さらにI/O発行回数を削減するために、複数のトランザクションを一括して書き出すために、しばらくの間スリープし、その間にログライトバッファへ複数トランザクションのログを溜める。

【0248】f) 規定時間後には自分で起きて、処理を繰り返す。しかし、他要因によって突然起こされて処理を始める場合もある。「その他の要因については次の項(2.5.2)で述べる。

(2.5.2)動作契機

ログライトデーモン105は以下の契機でログライトバッファの内容をログボリュームへ書き出す。

◎一定周期

ログライトデーモン105は一定周期毎にスリープ状態から自発的に目覚め、ログライトバッファ150の内容をログボリューム120へ反映する。

◎同期書き込み要求のトランザクション

トランザクションによっては同期書き込み要求で呼ばれる場合がある。このトランザクションについては、ログの書き出しを待たなければ終了してはならない。そのため、LOG__END呼び出し後に、明示的にLOG__SYNCを行う。LOG__SYNCはログライトデーモン105が寝ている場合には起こし、ログライトバッファの内容をその場で書き出させる。

【0249】現在ログライトデーモン105がI/O中であつたら、ログライトデーモン105の処理が終わる

のを寝て待つ。

◎ログライトバッファ不足

周期毎にログライトバッファをクリアしても、大きなログバッファが連続して終了すると、その前にログライトバッファがフルに近い状態になることがありえる。この時にはログライトデーモン105が起こされ、ログライトバッファ150の内容をログボリューム120に書き出してクリアする。

【0250】具体的には、あるトランザクションが自ログバッファの内容をLOG_ENDによってコピーしようとした時、ログライトバッファ150の残りサイズが自ログよりも小さいと判断された時、ログライトデーモン105を起こし、追加モードのログライトバッファ150を切り替えてもらい、その間は寝て待つ。

【0251】現在I/O中であり、かつ追加モードのログライトバッファ150が足りなくなった時には、そのトランザクションは待ち合わせざるを得ない。

◎メタキャッシュ不足

トランザクション実行中に、メタキャッシュ130域のいずれかのメタデータを追い出さなければ必要なメタデータをメタボリューム111～113から読み込むことができずに処理が進まなくなる場合が考えられる。

【0252】そのため、トランザクションが現在キャッシュ上のメタデータを追い出そうとする時、メタキャッシュ130上のメタデータが全てダーティであり、かつ、ログ待ちリストに繋がれたメタデータが存在する場合には、ログライトデーモン105を起動する。

◎アンマウント時

アンマウント時には必ず書き出さなければならない。そのため、アンマウント処理の延長でログライトデーモン105を起こし、書き出しを行う。この時、アンマウントを実行するため、該当ファイルシステムにメタデータを更新するようなトランザクションは動作していないことが保証される。従って、現在のログライトバッファの内容を書き出せば、それで全てである。

【0253】ログライトとバッファの内容を書き出した後、ログライトデーモン105は終了する。

(3) メタデータ管理

次に、メタデータの管理方式について説明する。メタキャッシュ130内のメタデータはmetalist構造体によって管理される。metalist構造体には以下のメンバがある。

- ・メタデータへのポインタ
- ・TRANS (トランザクション) リストポインタ
- ・メタライトリストprevポインタ
- ・メタライトリストnextポインタ
- ・ログ待ちリストprevポインタ
- ・ログ待ちリストnextポインタ
- ・ログシーケンス番号
- ・ログボリューム内オフセット

- ・トランザクション番号
- ・状態フラグ
- ・メタデータタイプ
- ・buf構造体実体

(3.1) メタデータの状態遷移

metalist構造体には次の6種類の状態があり、4種類のリストに繋がれる。それはmetalist構造体のフラグによって示され、それぞれターミナルを別とするリストに繋がれることによって実現する。全てのmetalist構造体はいずれかのリストに繋がっており、例外はない。

A) 空き管理構造更新状態

データ空き領域を管理するメタデータがトランザクションによって更新されると、リストに繋ぎ、将来まとめてログバッファにコピーすることは既に述べた。トランザクション終了時に本リストに繋がれたメタデータが直接ログボリュームへ反映される。リストは並行動作数に応じて必要である。このリストをBTFリストと呼ぶ。

【0254】この状態にあるメタデータはまだログバッファへはコピーされておらず、同一トランザクションによって再度更新される場合がある。既にリストに繋がれているメタデータであった場合は、リスト状態は変更しない。当然、メタボリュームに反映してはならない。

【0255】BTFリストは単方向環状リストであり、トランザクション途中状態と同じメンバを用いて繋がれている。

B) 利用域管理構造更新状態

実施例ではファイルシステムをエクステント管理している。iノードから導かれる、そのファイルが利用しているエクステントを示す、間接エクステントブロックについても、トランザクション内で1つの間接エクステントブロックが何度も更新される場合が考えられる。そこで、空き管理構造の場合と同様に、トランザクション中は独自のリストに繋ぎ、トランザクション終了時にまとめてログバッファへコピーする。リストは並行動作数に応じて必要である。このリストをBTAリストと呼ぶ。

【0256】この状態にある間接エクステントブロックはまだログバッファへはコピーされておらず、同一トランザクションによって再度更新される場合がある。既にリストに繋がれている間接エクステントブロックであった場合は、リスト状態は変更しない。当然、メタボリュームに反映してはならない。

【0257】BTAリストは単方向環状リストであり、トランザクション途中状態と同じメンバを用いて繋がれている。

C) トランザクション途中状態

トランザクション途中を示す。この状態のとき、該当するメタデータを更新したトランザクションは、ある1つのログバッファを専有しており、既にそこに更新後状態がコピーされている。リストは並行動作数に応じて必要

である。このリストをTRANSリストと呼ぶ。

【0258】しかし、まだトランザクションが終了していないことからログがログボリューム120へ反映されておらず、本メタデータもメタボリュームへの反映はできない。この状態にあるメタデータはファイルのロックによって保護されているため、他トランザクションから参照・更新されることはない。

【0259】TRANSリストのターミナルは配列になっており、その要素番号はトランザクションが用いているログバッファの番号（トランザクション番号）に対応する。

【0260】メタライトリストやログ待ちリストに繋がれているメタデータが繋がる場合がある。TRANSリストは単方向環状リストであり、BTFリストが用いるメンバを併用する。

【0261】D) ログ待ち状態

トランザクションは既に終了しているが、ログライトバッファの内容はログライトデモン105によってログボリューム120へ反映されるため、この時点ではまだログボリューム120に反映されていない状態である。

（デモンは同期ライトを行うが、トランザクションの視点から見れば、ログ反映が非同期に行われているように見える。）

トランザクションが終了する際に、C) の状態にあるTRANSリスト（巨大トランザクションの場合はBTFリストも）をログ待ちリストに繋がかえる。このリストはトランザクションが終了する毎に後方へ伸びるリストであり、ログライトバッファと同数の2本存在する。

【0262】この状態にあるメタデータは既にトランザクションが終了しており、ログはログライトバッファにコピーされている。トランザクションが終了していることから、他トランザクションによって参照・更新される可能性がある。この時（他トランザクションによって状態が変化した時）、リスト位置は変更せず、状態フラグだけを変更する。

【0263】メタライトリストに繋がれたメタデータが、他トランザクションによって再度更新されたために、トランザクション途中状態になり、そのトランザクションが終了したことによって、ログ待ち状態になった場合には、このメタデータはメタライトリストにも繋がれている。

【0264】ログライトデモン105によって、ログの反映が進むと、それに応じたメタデータをメタライトリストへ追加していく。この時、既にメタライトリストに繋がれているメタデータは、そのままの位置でいなければならない。

E) 書き出し可能状態

これは、ログバッファのログボリュームへ反映が終了し、自由にメタデータをメタボリュームへ反映することができるようになった状態である。この状態にあるメタ

データは他トランザクションによって参照・更新される可能性がある。この時、リスト位置は変更せず、状態フラグだけを変更する。

【0265】繋がるリストはメタライトリストであり、1本だけ存在する。メタライトデモン104はこのリストを参照してメタボリュームへの反映を行う。

F) I/O中状態

この状態にあるメタデータは現在非同期ライトによってメタボリュームに対してI/Oを投げているが、まだ完了していない。I/O途中であるため、他トランザクションは操作することができない（参照することはできる）。メタライトリストにそのまま繋がれている。

（3-2）ログボリュームの上書き判定情報

ログボリュームはサイズが有限であるため、サイクリックに利用し、過去のログを上書きしてシステムは動作する。ここで、過去のログを上書きするためには、そのトランザクションが更新したメタデータが全てメタボリュームに反映されていることが必要である。上書き可能領域を管理するために、以下の構造を設け、メタライトデモン104が変更、ログライトデモン105が参照する。

◎ログシーケンス番号

各トランザクションが終了し、ログバッファの内容をログライトバッファへコピーする毎に与えられる、シーケンシャル番号である。ログボリューム内のBEGINマーク、ENDマークに含まれる番号と同一である。既にライトリストに繋がるメタデータが再度更新される場合にも、この番号は変更されない。

◎ログボリューム内オフセット

ログボリューム内にはトランザクション毎のログが連続して並んでおり、上書きするためには、それぞれのトランザクションが更新した全てのメタデータがメタボリュームへ反映されている必要がある。

【0266】同一のログ番号を持つmetalist構造体は、ここにはそのログの先頭オフセットを挿入する。LOG_ENDによってログバッファの内容がログライトバッファにコピーされる際、そのアドレスとログボリュームの書き出しオフセットから導かれる。トランザクション毎のログボリューム内オフセットがTRANSリストを辿りながら、トランザクション毎のログボリューム内オフセットをそれぞれのmetalist構造体に設定する。

【0267】ログライトデモン105は、ログを書き出す際にメタライトリストの先頭に繋がるmetalist構造体を参照し、現在のポインタにログライトバッファに入っているログサイズを足した位置が、このオフセット値を超えないことを確認した後で書き出しを行う。

（3.3）メタボリュームへの反映

トランザクションによって更新されたメタデータは、ト

ランザクションが終了しログがログボリューム120に反映されるまでは書き出されてはならない。そのために、メタキャッシュ130上の各メタデータはそれぞれの状態に応じてリストに繋がれ、唯一メタボリュームへの反映が許可されているリストはメタライトリストである。

【0268】時間のかかるI/O要求をトランザクション内で行うことを避けるために、メタデータのメタボリュームへの反映はトランザクションとは関係ないところで動作するデーモンに委ねる。このデーモンはメタライトリストだけを意識し、`metalist`構造体内の情報から状態に応じてI/O発行するかどうかを決定し、関連付けられた**buf**構造体を用いてメタデータをメタボリュームへ反映する。

【0269】デーモンはメタライトリストに繋がれたメタデータを書き出した後、そのメタデータをリストから外し、`clean`であると設定する。

(3.4) メタライトデーモン

メタライトデーモン104は、マウント時に起動され、アンマウント時に停止する。すなわち、ファイルシステム毎にスレッドを起動する。

【0270】ログ書き出しの終わったトランザクションが更新したメタデータは、それを管理する**metalist**構造体が全てメタライトリストに繋がれている。メタライトデーモン104はメタライトリストに繋がるメタデータを、ログボリュームの残りが少なくなったと判断された場合などに非同期でメタボリュームに反映する。反映している間は該当するメタデータは更新することができず、I/O終了を待ち合わせることになる。

【0271】メタライトデーモン104はメタライトリストを意識し、繋がるメタデータをメタボリュームへ反映する。これにより、ログボリュームの上書き可能領域が拡大する。

【0272】しかし、メタライトリストに繋がれているメタデータの中にも、再度トランザクションによって更新が進み、メタボリュームへの反映が禁じられているものが存在する場合がある。この時、他のメタデータを書き出すとしても、そのメタデータについては非同期ライトの発行を待ち合わせなければならない。このようなメタデータが存在すると、以降のメタデータを全て書き出せたとしても、上書き可能領域を拡大することができない。

【0273】メタライトデーモン104は他者から起動されて処理を開始する場合と、自発的に起動する場合がある。以下に処理手順を述べるが、システムの状態（ログボリュームの空きやメタキャッシュ130の空きなど）に応じて動作が若干異なる。また、ここで述べる処理手順にはデーモン内だけではなく、ドライバから呼ばれる関数での処理も含まれている。

(3.4.1) 自発起動処理

メタライトデーモン104はタイマによって自発的に起動して、それまでにメタライトリストに繋がれた書き出し可能なメタデータをメタボリュームに反映する。この時、メタライトリストの全てを書き出す訳ではなく、ある一定の数だけ非同期書き出しを行う。普段から少しずつメタデータの書き出しを行っておくことによって、資源不足による起動を避け、I/O負荷分散を図る目的がある。

【0274】a) 自発起動時には、メタライトリストに繋がるメタデータ数を調査する。

b) メタライトリストの先頭から、メタデータの状態を調べる。

c) メタデータが書き出し可能状態であれば、その状態をI/O中状態に変更し、非同期ライトを発行する。

【0275】d) `b_iodone`の関数がI/Oの成功を確認する。

e) `b_iodone`の関数をメタライトリストから外す。

f) 規定数だけ繰り返す

g) スリープする。

【0276】以下、詳細説明である。

a) 自発起動間隔として定義する間隔毎にメタライトリストに繋がるメタデータ数を確認する。具体的には、メタライトリストのターミナルに含まれる、リンクされたメタデータ数を調べる。このとき、それほど多くのメタデータが繋がれていない場合には書き出しは行わない。

【0277】b) メタライトリストには基本的にはログの書き出しが終わった、書き出し可能状態のメタデータが繋がれている。しかし、トランザクションが終了してメタライトリストに繋がれた後で他トランザクションによって更新されると、そのメタデータだけは書き出し不可の状態に戻ってしまう。そのため、メタライトデーモン104はメタライトリストに繋がるメタデータの状態を確認しながら書き出し処理を行わなければならない。

【0278】c) 現在ポイントするメタデータが書き出し可能状態であれば、その状態をI/O状態に書き換え、`metalist`構造体に含まれる**buf**構造体を用いてI/Oを発行する。I/O中状態のメタデータは他トランザクションから更新されることはない。メタデータのディスクライトは非同期ライトで行い、デーモンはI/Oの結果を待たずに次の処理へ進む。

【0279】メタデータが書き出し不可状態であるなら、そのメタデータは諦め、リストの次に繋がるメタデータに進み、状態を調べる。

d) 非同期ライトによって発行されたI/Oの結果を判定するのは、**buf**構造体のメンバ**b_iodone**に組み込まれた関数である。この関数には**buf**構造体が引数に与えられ、それを元にエラー判定処理を呼び出す。ここでエラーを検出した場合には、異常系処理へ進む(省略)。成功していた場合には、該当メタデータの

clean数をインクリメントする。

【0280】e) b__iodoneに組み込まれた関数はメタライトリストの管理も行う。引数に渡されたbuf構造体の上を見るとそこはmetalist構造体になっており、そのフラグからI/O中フラグを落とし、メタライトリストから外す。また、メタデータをダーティ状態からclean状態へ変更する。これは、メタキャッシュ130で管理されるメタデータである場合には、それぞれの管理構造体で管理されており、この管理構造体はmetalist構造体からポイントされる。

【0281】全ての処理が終了したら、このmetalist構造体をリリースする。

f) メタライトリストに繋がる書き出し可能メタデータを、次々とリストを辿りながら定義した数だけ処理を繰り返す。ここで、書き出しが不可とされているメタデータについては、この数には含めない。また、メタライトリストが定義数よりも少ない場合には、当然そこで終了となる。

g) スリープする。再度、自発的に起動するか、あるいは資源不足によって他から起動されるまで、スリープは継続する。

(3.4.2) 平常処理

システム状態が以下の場合には、ここで述べる処理手順に従いメタライトデーモン104は動作する。

◎ログボリュームの残りが少ない。

【0282】ログライトデーモン105が判断する。ログライトバッファを書き出す際に、上書き可能域の大きさを計算し、これが所定の閾値を下回った時にメタライトデーモン104を起動する。

◎メタキャッシュの残りが少ない。

【0283】更新リリースがあった時、各メタデータのclean数がデクリメントされるが、その数が所定の閾値を下回った時にメタライトデーモン104を起動する。

a) メタライトリストの先頭から、メタデータの状態を調べる。

【0284】b) メタデータが書き出し可能状態であれば、その状態をI/O中状態に変更し、非同期ライトを発行する。

c) b__iodoneの関数がI/Oの成功を確認する。

【0285】d) b__iodoneの関数をメタライトリストから外す。

e) メタライトリストの最後まで繰り返す。

f) スリープする。

【0286】以下、詳細説明である。

a) ~ d) 自発的に起動した場合と同じである。

e) メタライトリストを辿りながら、繋がる全ての書き出し可能メタデータについて処理を繰り返す。平常処理時には、書き出し不可のメタデータについては飛ばして

処理を行う。そのため、ログボリューム不足時には、メタライトリストの先頭、すなわち、ログボリュームの上書き可能位置を制限しているメタデータが書き出せなければ資源不足は解消しないが、平常処理であるため、ここでは特別な対処を行わない。

【0287】f) タイマを設定してスリープする。

(3.4.3) 緊急処理

システム状態が以下の場合には、ここで述べる処理手順に従いメタライトデーモン104は動作する。

◎ログボリュームの残りが非常に少ない。

◎メタキャッシュ130の残りが非常に少ない。

【0288】a) トランザクションの新規開始を制限する。

b) メタライトリストの先頭から、メタデータの状態を調べる。

c) メタデータが書き出し可能状態であれば、その状態をI/O中状態に変更し、非同期ライトを発行する。

【0289】d) ログ未反映状態となっているメタデータがあれば、ログライトデーモン105を起動する。

e) b__iodoneの関数がI/Oの成功を確認する。

【0290】f) b__iodoneの関数をメタライトリストから外す。

g) 繰り返す。

h) 現在の資源状態を調査し、不足があれば、それが解消するまで繰り返す。

【0291】i) トランザクションの受付を開始する。

j) スリープする。

以下、詳細説明である。

【0292】a) 緊急時には、新規のトランザクションの開始を受け付けない。具体的には、トランザクションに利用するログバッファを与えないことによって、そのトランザクションのBEGIN宣言時にスリープさせる。そのために、特定のフラグを設け、BEGIN宣言時にそのフラグを参照しなければならない。

【0293】b) ~ g) ここは基本的に平常処理の場合と同じ処理である。すなわち、メタライトリストの先頭から、追い出し不可状態のメタデータは飛ばして、書き出し可能状態のメタデータを順に非同期ライトによって書き出す。

【0294】ただし、d) だけ異なる。

d) ログボリュームへの書き出しがデーモンによって行われることから、たとえトランザクションが終了していてもログが未反映のためにメタボリュームへの反映を拒否しているメタデータが存在することが考えられる。この場合は、強制的にログライトデーモン105を起動して、書き出し可能状態へ移行するように操作する。既にログライトデーモン105が動作中であれば、そのまま先へ進む。

【0295】h) ここで、現在の資源状態を調べる。平

常処理の動作契機となる閾値以上の資源が回復していなければ、再度、メタライトリスト内のメタデータ書き出しを試みる。ここで、d) によってログ未反映状態のメタデータが、書き出し可能となったことによって進展することを期待している。複数回、繰り返すことによって、新規トランザクションの受付を制限していることから、資源回復までメタデータの書き出しが可能であると考える。

【0296】 i) a) にて制限していたトランザクションの受付を再開する。

j) タイマを設定して、スリープする。

(4) 獲得解放処理

メタデータの割り当て状況を管理するビットマップの状態として、FREE-dirtyとALLOC-dirtyを区分し、FREE-dirtyなビットマップからは獲得しないようにする。

【0297】 具体的には以下の通りである。

- ・マウント時にビットマップを読み込む際に、1つを獲得用と定める。説明を簡単にするため、複数読み込むビットマップのうち、一番若いもの(最初に読み込むもの)を最初の獲得用ビットマップとする。

- ・獲得用ビットマップの複製を作成する。

- ・獲得時には複製を検索して獲得位置を決定し、本体と複製の両方のビットを操作する。

- ・解放時には本体のみのビットを操作する。

- ・複製ビットマップには解放が記録されないため、獲得処理が進むと全てのビットが立ち、そのビットマップでは獲得できないと判断される。その場合、現在メモリ上にあるビットマップのうち、CLEANなもの、またはALLOC-dirtyのビットマップを次の獲得用ビットマップと定義し、その複製を作成する。

- ・メモリ上のビットマップが全てFREE-dirtyである場合には、どれかを追い出し、新しいビットマップを獲得用ビットマップとして読み込む。そして、その複製を作成する。ここで、追い出し・読み込みのアルゴリズムは従来通りで良い。

【0298】 獲得用ビットマップとして選ばれたビットマップは追い出しの対象とはしない。そのためメタキャッシュ域不足によって他から追い出されることはない。また、メタライトリストに繋がったことによって、メタボリュームに反映された場合にも、CLEANな状態にはなるが、複製は更新せず、そのままとする。

【0299】 本実施の形態による効果は、以下の通りである。以上説明したように、本発明によれば複数の二次記憶装置に保存されたメタデータのログにボリューム情報を含め、また、有効なログの位置を算出・保存することでリプレイするログ量を減少せしめ、さらに、ログボリューム全体のゼロクリアをする必要をなくすことにより、ログ機構の最大の利点であるところのファイルシステム復元時間の短縮に及ぼす影響を小さくし、オーバーオールコンピュータシステム可用性の増大に寄与するところ

ろが大きい。

【0300】 さらに、本発明によれば一回トランザクション内で複数回更新されるメタデータのログを一度しか採取せず、また、ログバッファの分割や、獲得・解放処理の特殊なログ採取方式によって複数のトランザクションの独立性を考慮し、かつ、ログ機構導入による速度性能の劣化を最小に留めることにおいて寄与するところが大きい。

【0301】 加えて、本発明によればトランザクション毎に異なるログの採取量を考慮し、多くのメタデータを更新するトランザクションについては分割し、トランザクションに与えられたパラメタをもログに採取し、リプレイ時にそのパラメタを元に、途中で終わったトランザクションを再度実行することによって、オペレーションのセマンティクスを保証した復元が可能となる面において寄与するところが大きい。

【0302】 なお、上記の処理機能は、コンピュータによって実現することができる。その場合、説明した処理内容は、コンピュータで読み取り可能な記録媒体に記録されたプログラムに記述されており、このプログラムをコンピュータで実行することにより、上記処理がコンピュータで実現される。コンピュータで読み取り可能な記録媒体としては、磁気記録装置や半導体メモリ等がある。市場へ流通させる場合には、CD-ROM(Compact Disk Read Only Memory)やフロッピーディスク等の可搬型記録媒体にプログラムを格納して流通させたり、ネットワークを介して接続されたコンピュータの記憶装置に格納しておき、ネットワークを通じて他のコンピュータに転送することもできる。コンピュータで実行する際には、コンピュータ内のハードディスク装置等にプログラムを格納しておき、メインメモリにロードして実行する。

【0303】

【発明の効果】 以上説明したように、第1の発明では、メタキャッシュ内のメタデータとともにメタデータがどのメタボリュームから取り出されたのかを示すメタデータ管理情報をログとして採取するようにしたため、保持されたログがどのメタボリュームのメタデータに関するログであるのかを管理することができる。その結果、複数のログボリュームにメタデータが格納されていても、ファイルシステムの不整合を修正することが可能となる。

【0304】 また、第2の発明では、ログデータの有効範囲を管理するようにしたため、ファイルシステムを復元する際には、ログボリューム12内の有効なログのみを用いて効率よく復元処理を行うことが可能となる。

【0305】 また、第3の発明では、ログデータに対して付与するシーケンス番号の最大値を、システムの使用可能年数以上使い続けることができる値としたことで、常に昇順の採番が可能となり、ログボリュームのゼロク

リアに伴う処理の遅延を避けることができる。

【0306】また、第4の発明では、メタデータが複数回更新される場合には、最終形態のみをログとして採取するようにしたため、ログデータが短縮されるとともに、ログデータの短縮に伴いファイルシステム復元時間の短縮が図れる。

【0307】また、第5の発明では、割り当て管理情報の一部の複製を獲得操作管理情報とし、メタデータの獲得時には獲得操作管理情報内から取得すべきメタデータを特定するが、解放時には獲得操作管理情報の情報を更新しないようにしたため、メタデータの解放直後に別のトランザクションにより獲得されることがなくなる。その結果、システムダウン時に中途までしか終了していなかった解放トランザクションが解放したはずである領域は、解放される直前の状態のまま保全されることが保証される。

【0308】また、第6の発明では、獲得、解放操作のログとして、その操作対象となった情報のみを記録するようにしたため、ログ採取量が少量ですむ。また、第7の発明では、複数のログバッファを設け、さらにいくつか異なるサイズのログバッファを用意し、トランザクション毎のログをそのトランザクションに適したサイズのログバッファに格納するようにしたため、複数のトランザクションが並列実行される際のトランザクションの独立性を高めることができ、また、メモリ空間を有効に活用できる。

【0309】また、第8の発明では、1つのトランザクションのログがログバッファに収まらない場合に、中間ログとして完結されたログをログボリュームに書き出すようにしたため、部分的であるログを、ファイル復元時には1つのトランザクションのログと同値に扱うことが可能となり、ファイルシステムの復元時に望ましい状態に復元するのが容易となる。

【0310】また、第9の発明では、ログ採取に関するシステムの状態よってトランザクションの受け入れを制限するようにしたため、複数のトランザクションが並列実行されることによるメモリ枯渇等の障害の発生を防止することができる。

【図面の簡単な説明】

【図1】第1の発明の原理構成図である。

【図2】第2の発明の原理構成図である。

【図3】第3の発明の原理構成図である。

【図4】第4の発明の原理構成図である。

【図5】第5の発明の原理構成図である。

【図6】第6の発明の原理構成図である。

【図7】第7の発明の原理構成図である。

【図8】第8の発明の原理構成図である。

【図9】第9の発明の原理構成図である。

【図10】本発明を適用するデータ処理装置のハードウェア構成図である。

【図11】ファイルシステム上で動作するログ採取機能の構成図である。

【図12】メタデータ管理情報を示す図である。

【図13】ログバッファの形式を示す図である。

【図14】有効範囲を説明する図である。

【図15】ログ採取処理のフローチャートである。

【図16】メタボリュームの割り当て管理状況を示す図である。

【図17】ビットマップによるメタデータ獲得処理を示すフローチャートである。

【図18】解放処理のフローチャートである。

【図19】トランザクションの処理の開始と終了の状況を示す図である。

【図20】ログバッファへのログの格納状況を示す図である。

【図21】ログ採取手順を示すフローチャートである。

【図22】ファイルシステム復元処理を示すフローチャートである。

【図23】新規トランザクションの受け入れ許否判定処理を示すフローチャートである。

【符号の説明】

- 1 a～1 c メタボリューム
- 2 ログボリューム
- 3 メタキャッシュ
- 4 メタデータ読み込み手段
- 5 メタデータ管理情報
- 6 トランザクション
- 7 ログ採取手段
- 8 ログバッファ
- 9 ログ書き込み手段

【手続補正2】

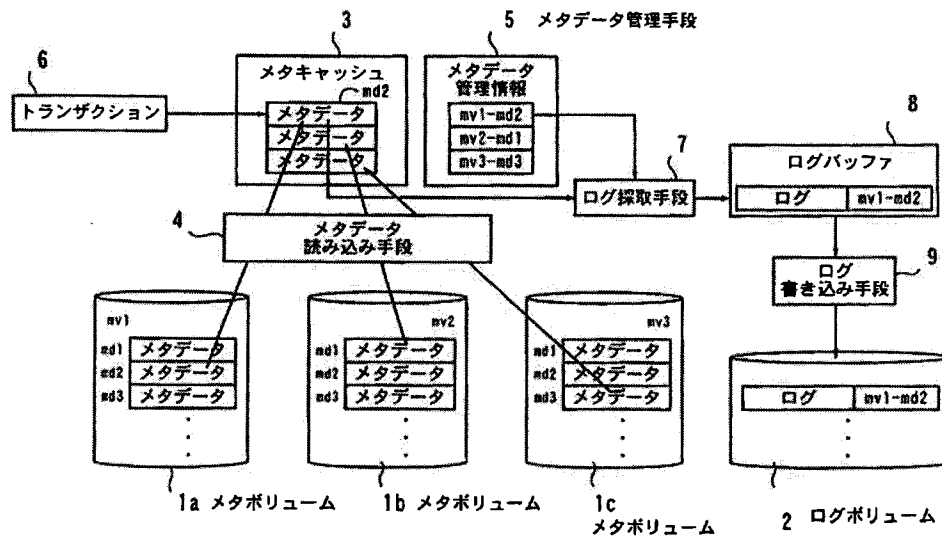
【補正対象書類名】図面

【補正対象項目名】図1

【補正方法】変更

【補正内容】

【図1】



【手続補正3】

【補正対象書類名】図面

【補正対象項目名】図23

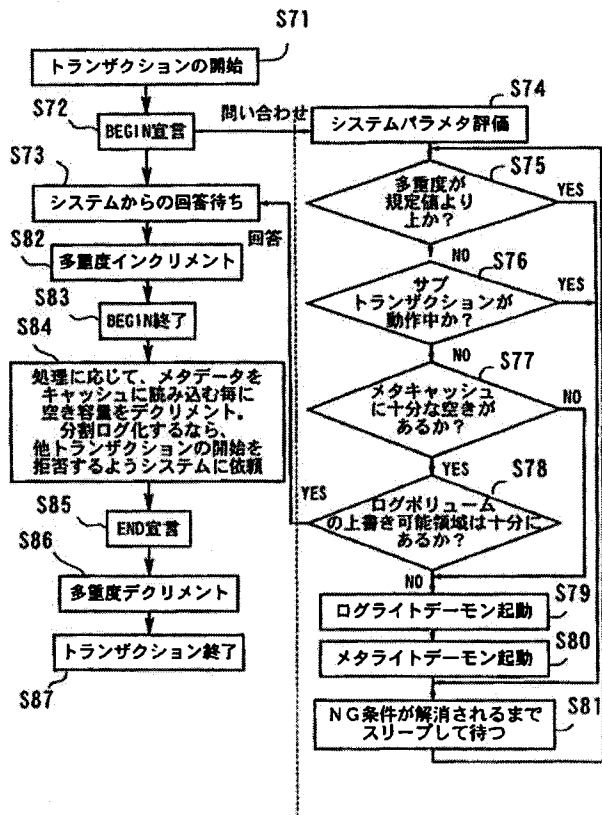
【補正方法】変更

【補正内容】

【図23】

トランザクションの動作

システムの動作



フロントページの続き

(72)発明者 鷺見 昌司
神奈川県川崎市中原区上小田中4丁目1番
1号 富士通株式会社内
(72)発明者 山口 悟
神奈川県川崎市中原区上小田中4丁目1番
1号 富士通株式会社内

(72)発明者 谷脇 貞善
神奈川県川崎市中原区上小田中4丁目1番
1号 富士通株式会社内
(72)発明者 浜中 征志郎
神奈川県川崎市中原区上小田中4丁目1番
1号 富士通株式会社内
Fターム(参考) 5B082 CA17 DC06 DC12 DD04 EA02
FA02 FA12 GB05 GB06